

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()	27

5.3.4.3 dump()	27
5.4 Tang::AstNodeBinary Class Reference	27
5.4.1 Detailed Description	30
5.4.2 Member Enumeration Documentation	30
5.4.2.1 Operation	30
5.4.2.2 PreprocessState	31
5.4.3 Constructor & Destructor Documentation	31
5.4.3.1 AstNodeBinary()	31
5.4.4 Member Function Documentation	31
5.4.4.1 compile()	31
5.4.4.2 compilePreprocess()	32
5.4.4.3 dump()	32
5.5 Tang::AstNodeBlock Class Reference	33
5.5.1 Detailed Description	35
5.5.2 Member Enumeration Documentation	35
5.5.2.1 PreprocessState	35
5.5.3 Constructor & Destructor Documentation	35
5.5.3.1 AstNodeBlock()	35
5.5.4 Member Function Documentation	36
5.5.4.1 compile()	36
5.5.4.2 compilePreprocess()	36
5.5.4.3 dump()	37
5.6 Tang::AstNodeBoolean Class Reference	37
5.6.1 Detailed Description	39
5.6.2 Member Enumeration Documentation	39
5.6.2.1 PreprocessState	39
5.6.3 Constructor & Destructor Documentation	39
5.6.3.1 AstNodeBoolean()	39
5.6.4 Member Function Documentation	40
5.6.4.1 compile()	40
5.6.4.2 compilePreprocess()	40
5.6.4.3 dump()	41
5.7 Tang::AstNodeBreak Class Reference	41
5.7.1 Detailed Description	43
5.7.2 Member Enumeration Documentation	43
5.7.2.1 PreprocessState	43
5.7.3 Constructor & Destructor Documentation	43
5.7.3.1 AstNodeBreak()	43
5.7.4 Member Function Documentation	43
5.7.4.1 compile()	44
5.7.4.2 compilePreprocess()	44
5.7.4.3 dump()	45

5.8 Tang::AstNodeCast Class Reference	45
5.8.1 Detailed Description	47
5.8.2 Member Enumeration Documentation	47
5.8.2.1 PreprocessState	47
5.8.2.2 Type	47
5.8.3 Constructor & Destructor Documentation	48
5.8.3.1 AstNodeCast()	48
5.8.4 Member Function Documentation	48
5.8.4.1 compile()	48
5.8.4.2 compilePreprocess()	49
5.8.4.3 dump()	49
5.9 Tang::AstNodeContinue Class Reference	50
5.9.1 Detailed Description	52
5.9.2 Member Enumeration Documentation	52
5.9.2.1 PreprocessState	52
5.9.3 Constructor & Destructor Documentation	52
5.9.3.1 AstNodeContinue()	52
5.9.4 Member Function Documentation	52
5.9.4.1 compile()	53
5.9.4.2 compilePreprocess()	53
5.9.4.3 dump()	54
5.10 Tang::AstNodeDoWhile Class Reference	54
5.10.1 Detailed Description	56
5.10.2 Member Enumeration Documentation	56
5.10.2.1 PreprocessState	56
5.10.3 Constructor & Destructor Documentation	56
5.10.3.1 AstNodeDoWhile()	56
5.10.4 Member Function Documentation	57
5.10.4.1 compile()	57
5.10.4.2 compilePreprocess()	57
5.10.4.3 dump()	58
5.11 Tang::AstNodeFloat Class Reference	58
5.11.1 Detailed Description	60
5.11.2 Member Enumeration Documentation	60
5.11.2.1 PreprocessState	60
5.11.3 Constructor & Destructor Documentation	60
5.11.3.1 AstNodeFloat()	60
5.11.4 Member Function Documentation	61
5.11.4.1 compile()	61
5.11.4.2 compilePreprocess()	61
5.11.4.3 dump()	62
5.12 Tang::AstNodeFor Class Reference	62

5.12.1 Detailed Description	64
5.12.2 Member Enumeration Documentation	64
5.12.2.1 PreprocessState	64
5.12.3 Constructor & Destructor Documentation	65
5.12.3.1 AstNodeFor()	65
5.12.4 Member Function Documentation	65
5.12.4.1 compile()	65
5.12.4.2 compilePreprocess()	66
5.12.4.3 dump()	66
5.13 Tang::AstNodeFunctionCall Class Reference	67
5.13.1 Detailed Description	68
5.13.2 Member Enumeration Documentation	68
5.13.2.1 PreprocessState	68
5.13.3 Constructor & Destructor Documentation	69
5.13.3.1 AstNodeFunctionCall()	69
5.13.4 Member Function Documentation	69
5.13.4.1 compile()	69
5.13.4.2 compilePreprocess()	70
5.13.4.3 dump()	70
5.14 Tang::AstNodeFunctionDeclaration Class Reference	70
5.14.1 Detailed Description	72
5.14.2 Member Enumeration Documentation	72
5.14.2.1 PreprocessState	72
5.14.3 Constructor & Destructor Documentation	72
5.14.3.1 AstNodeFunctionDeclaration()	72
5.14.4 Member Function Documentation	73
5.14.4.1 compile()	73
5.14.4.2 compilePreprocess()	73
5.14.4.3 dump()	74
5.15 Tang::AstNodeIdentifier Class Reference	74
5.15.1 Detailed Description	76
5.15.2 Member Enumeration Documentation	76
5.15.2.1 PreprocessState	76
5.15.3 Constructor & Destructor Documentation	77
5.15.3.1 AstNodeIdentifier()	77
5.15.4 Member Function Documentation	77
5.15.4.1 compile()	77
5.15.4.2 compilePreprocess()	78
5.15.4.3 dump()	78
5.16 Tang::AstNodeIfElse Class Reference	79
5.16.1 Detailed Description	81
5.16.2 Member Enumeration Documentation	81

5.16.2.1 PreprocessState	81
5.16.3 Constructor & Destructor Documentation	82
5.16.3.1 AstNodeIfElse() [1/2]	82
5.16.3.2 AstNodeIfElse() [2/2]	82
5.16.4 Member Function Documentation	82
5.16.4.1 compile()	82
5.16.4.2 compilePreprocess()	83
5.16.4.3 dump()	83
5.17 Tang::AstNodeIndex Class Reference	84
5.17.1 Detailed Description	86
5.17.2 Member Enumeration Documentation	86
5.17.2.1 PreprocessState	86
5.17.3 Constructor & Destructor Documentation	86
5.17.3.1 AstNodeIndex()	87
5.17.4 Member Function Documentation	87
5.17.4.1 compile()	87
5.17.4.2 compilePreprocess()	88
5.17.4.3 dump()	88
5.17.4.4 getCollection()	88
5.17.4.5 getIndex()	89
5.18 Tang::AstNodeInteger Class Reference	89
5.18.1 Detailed Description	91
5.18.2 Member Enumeration Documentation	91
5.18.2.1 PreprocessState	91
5.18.3 Constructor & Destructor Documentation	91
5.18.3.1 AstNodeInteger()	91
5.18.4 Member Function Documentation	92
5.18.4.1 compile()	92
5.18.4.2 compilePreprocess()	92
5.18.4.3 dump()	93
5.19 Tang::AstNodeMap Class Reference	93
5.19.1 Detailed Description	94
5.19.2 Member Enumeration Documentation	94
5.19.2.1 PreprocessState	94
5.19.3 Constructor & Destructor Documentation	95
5.19.3.1 AstNodeMap()	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::AstNodePrint Class Reference	97
5.20.1 Detailed Description	99

5.20.2 Member Enumeration Documentation	99
5.20.2.1 PreprocessState	99
5.20.2.2 Type	99
5.20.3 Constructor & Destructor Documentation	100
5.20.3.1 AstNodePrint()	100
5.20.4 Member Function Documentation	100
5.20.4.1 compile()	100
5.20.4.2 compilePreprocess()	101
5.20.4.3 dump()	101
5.21 Tang::AstNodeRangedFor Class Reference	102
5.21.1 Detailed Description	103
5.21.2 Member Enumeration Documentation	103
5.21.2.1 PreprocessState	103
5.21.3 Constructor & Destructor Documentation	103
5.21.3.1 AstNodeRangedFor()	103
5.21.4 Member Function Documentation	104
5.21.4.1 compile()	104
5.21.4.2 compilePreprocess()	105
5.21.4.3 dump()	105
5.22 Tang::AstNodeReturn Class Reference	106
5.22.1 Detailed Description	108
5.22.2 Member Enumeration Documentation	108
5.22.2.1 PreprocessState	108
5.22.3 Constructor & Destructor Documentation	108
5.22.3.1 AstNodeReturn()	108
5.22.4 Member Function Documentation	109
5.22.4.1 compile()	109
5.22.4.2 compilePreprocess()	109
5.22.4.3 dump()	110
5.23 Tang::AstNodeSlice Class Reference	110
5.23.1 Detailed Description	112
5.23.2 Member Enumeration Documentation	112
5.23.2.1 PreprocessState	112
5.23.3 Constructor & Destructor Documentation	113
5.23.3.1 AstNodeSlice()	113
5.23.4 Member Function Documentation	113
5.23.4.1 compile()	113
5.23.4.2 compilePreprocess()	114
5.23.4.3 dump()	114
5.24 Tang::AstNodeString Class Reference	115
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 AstNodeString()	117
5.24.4 Member Function Documentation	118
5.24.4.1 compile()	118
5.24.4.2 compileLiteral()	118
5.24.4.3 compilePreprocess()	119
5.24.4.4 dump()	119
5.25 Tang::AstNodeTernary Class Reference	120
5.25.1 Detailed Description	122
5.25.2 Member Enumeration Documentation	122
5.25.2.1 PreprocessState	122
5.25.3 Constructor & Destructor Documentation	122
5.25.3.1 AstNodeTernary()	123
5.25.4 Member Function Documentation	123
5.25.4.1 compile()	123
5.25.4.2 compilePreprocess()	124
5.25.4.3 dump()	124
5.26 Tang::AstNodeUnary Class Reference	124
5.26.1 Detailed Description	126
5.26.2 Member Enumeration Documentation	126
5.26.2.1 Operator	126
5.26.2.2 PreprocessState	127
5.26.3 Constructor & Destructor Documentation	127
5.26.3.1 AstNodeUnary()	127
5.26.4 Member Function Documentation	127
5.26.4.1 compile()	127
5.26.4.2 compilePreprocess()	129
5.26.4.3 dump()	129
5.27 Tang::AstNodeWhile Class Reference	130
5.27.1 Detailed Description	132
5.27.2 Member Enumeration Documentation	132
5.27.2.1 PreprocessState	132
5.27.3 Constructor & Destructor Documentation	132
5.27.3.1 AstNodeWhile()	132
5.27.4 Member Function Documentation	133
5.27.4.1 compile()	133
5.27.4.2 compilePreprocess()	134
5.27.4.3 dump()	134
5.28 Tang::ComputedExpression Class Reference	134
5.28.1 Detailed Description	136
5.28.2 Member Function Documentation	137

5.28.2.1	<code>__add()</code>	137
5.28.2.2	<code>__asCode()</code>	137
5.28.2.3	<code>__assign_index()</code>	137
5.28.2.4	<code>__boolean()</code>	138
5.28.2.5	<code>__divide()</code>	138
5.28.2.6	<code>__equal()</code>	138
5.28.2.7	<code>__float()</code>	139
5.28.2.8	<code>__getIterator()</code>	139
5.28.2.9	<code>__index()</code>	139
5.28.2.10	<code>__integer()</code>	140
5.28.2.11	<code>__iteratorNext()</code>	140
5.28.2.12	<code>__lessThan()</code>	140
5.28.2.13	<code>__modulo()</code>	141
5.28.2.14	<code>__multiply()</code>	141
5.28.2.15	<code>__negative()</code>	142
5.28.2.16	<code>__not()</code>	142
5.28.2.17	<code>__slice()</code>	142
5.28.2.18	<code>__string()</code>	143
5.28.2.19	<code>__subtract()</code>	143
5.28.2.20	<code>dump()</code>	143
5.28.2.21	<code>is_equal()</code> [1/6]	144
5.28.2.22	<code>is_equal()</code> [2/6]	144
5.28.2.23	<code>is_equal()</code> [3/6]	145
5.28.2.24	<code>is_equal()</code> [4/6]	145
5.28.2.25	<code>is_equal()</code> [5/6]	145
5.28.2.26	<code>is_equal()</code> [6/6]	146
5.28.2.27	<code>isCopyNeeded()</code>	146
5.28.2.28	<code>makeCopy()</code>	147
5.29	Tang::ComputedExpressionArray Class Reference	147
5.29.1	Detailed Description	149
5.29.2	Constructor & Destructor Documentation	150
5.29.2.1	<code>ComputedExpressionArray()</code>	150
5.29.3	Member Function Documentation	150
5.29.3.1	<code>__add()</code>	150
5.29.3.2	<code>__asCode()</code>	150
5.29.3.3	<code>__assign_index()</code>	151
5.29.3.4	<code>__boolean()</code>	151
5.29.3.5	<code>__divide()</code>	151
5.29.3.6	<code>__equal()</code>	152
5.29.3.7	<code>__float()</code>	152
5.29.3.8	<code>__getIterator()</code>	152
5.29.3.9	<code>__index()</code>	153

5.29.3.10	<code>__integer()</code>	153
5.29.3.11	<code>__iteratorNext()</code>	154
5.29.3.12	<code>__lessThan()</code>	154
5.29.3.13	<code>__modulo()</code>	154
5.29.3.14	<code>__multiply()</code>	155
5.29.3.15	<code>__negative()</code>	155
5.29.3.16	<code>__not()</code>	155
5.29.3.17	<code>__slice()</code>	156
5.29.3.18	<code>__string()</code>	156
5.29.3.19	<code>__subtract()</code>	157
5.29.3.20	<code>dump()</code>	157
5.29.3.21	<code>is_equal()</code> [1/6]	158
5.29.3.22	<code>is_equal()</code> [2/6]	158
5.29.3.23	<code>is_equal()</code> [3/6]	158
5.29.3.24	<code>is_equal()</code> [4/6]	159
5.29.3.25	<code>is_equal()</code> [5/6]	159
5.29.3.26	<code>is_equal()</code> [6/6]	159
5.29.3.27	<code>isCopyNeeded()</code>	160
5.29.3.28	<code>makeCopy()</code>	160
5.30	Tang::ComputedExpressionBoolean Class Reference	161
5.30.1	Detailed Description	163
5.30.2	Constructor & Destructor Documentation	163
5.30.2.1	<code>ComputedExpressionBoolean()</code>	163
5.30.3	Member Function Documentation	163
5.30.3.1	<code>__add()</code>	163
5.30.3.2	<code>__asCode()</code>	164
5.30.3.3	<code>__assign_index()</code>	164
5.30.3.4	<code>__boolean()</code>	164
5.30.3.5	<code>__divide()</code>	164
5.30.3.6	<code>__equal()</code>	165
5.30.3.7	<code>__float()</code>	165
5.30.3.8	<code>__getIterator()</code>	166
5.30.3.9	<code>__index()</code>	166
5.30.3.10	<code>__integer()</code>	166
5.30.3.11	<code>__iteratorNext()</code>	166
5.30.3.12	<code>__lessThan()</code>	167
5.30.3.13	<code>__modulo()</code>	167
5.30.3.14	<code>__multiply()</code>	168
5.30.3.15	<code>__negative()</code>	168
5.30.3.16	<code>__not()</code>	168
5.30.3.17	<code>__slice()</code>	168
5.30.3.18	<code>__string()</code>	169

5.30.3.19	<code>__subtract()</code>	169
5.30.3.20	<code>dump()</code>	170
5.30.3.21	<code>is_equal()</code> [1/6]	170
5.30.3.22	<code>is_equal()</code> [2/6]	170
5.30.3.23	<code>is_equal()</code> [3/6]	171
5.30.3.24	<code>is_equal()</code> [4/6]	171
5.30.3.25	<code>is_equal()</code> [5/6]	171
5.30.3.26	<code>is_equal()</code> [6/6]	172
5.30.3.27	<code>isCopyNeeded()</code>	172
5.30.3.28	<code>makeCopy()</code>	172
5.31	Tang::ComputedExpressionCompiledFunction Class Reference	173
5.31.1	Detailed Description	175
5.31.2	Constructor & Destructor Documentation	175
5.31.2.1	<code>ComputedExpressionCompiledFunction()</code>	175
5.31.3	Member Function Documentation	175
5.31.3.1	<code>__add()</code>	175
5.31.3.2	<code>__asCode()</code>	176
5.31.3.3	<code>__assign_index()</code>	176
5.31.3.4	<code>__boolean()</code>	176
5.31.3.5	<code>__divide()</code>	177
5.31.3.6	<code>__equal()</code>	177
5.31.3.7	<code>__float()</code>	178
5.31.3.8	<code>__getIterator()</code>	178
5.31.3.9	<code>__index()</code>	178
5.31.3.10	<code>__integer()</code>	179
5.31.3.11	<code>__iteratorNext()</code>	179
5.31.3.12	<code>__lessThan()</code>	179
5.31.3.13	<code>__modulo()</code>	180
5.31.3.14	<code>__multiply()</code>	180
5.31.3.15	<code>__negative()</code>	181
5.31.3.16	<code>__not()</code>	181
5.31.3.17	<code>__slice()</code>	181
5.31.3.18	<code>__string()</code>	182
5.31.3.19	<code>__subtract()</code>	182
5.31.3.20	<code>dump()</code>	182
5.31.3.21	<code>is_equal()</code> [1/6]	183
5.31.3.22	<code>is_equal()</code> [2/6]	183
5.31.3.23	<code>is_equal()</code> [3/6]	183
5.31.3.24	<code>is_equal()</code> [4/6]	184
5.31.3.25	<code>is_equal()</code> [5/6]	184
5.31.3.26	<code>is_equal()</code> [6/6]	184
5.31.3.27	<code>isCopyNeeded()</code>	185

5.31.3.28 makeCopy()	185
5.32 Tang::ComputedExpressionError Class Reference	186
5.32.1 Detailed Description	189
5.32.2 Constructor & Destructor Documentation	189
5.32.2.1 ComputedExpressionError()	189
5.32.3 Member Function Documentation	189
5.32.3.1 __add()	189
5.32.3.2 __asCode()	189
5.32.3.3 __assign_index()	190
5.32.3.4 __boolean()	190
5.32.3.5 __divide()	190
5.32.3.6 __equal()	191
5.32.3.7 __float()	191
5.32.3.8 __getIterator()	191
5.32.3.9 __index()	192
5.32.3.10 __integer()	192
5.32.3.11 __iteratorNext()	192
5.32.3.12 __lessThan()	193
5.32.3.13 __modulo()	193
5.32.3.14 __multiply()	194
5.32.3.15 __negative()	194
5.32.3.16 __not()	194
5.32.3.17 __slice()	194
5.32.3.18 __string()	195
5.32.3.19 __subtract()	195
5.32.3.20 dump()	196
5.32.3.21 is_equal() [1/6]	196
5.32.3.22 is_equal() [2/6]	196
5.32.3.23 is_equal() [3/6]	197
5.32.3.24 is_equal() [4/6]	197
5.32.3.25 is_equal() [5/6]	197
5.32.3.26 is_equal() [6/6]	198
5.32.3.27 isCopyNeeded()	198
5.32.3.28 makeCopy()	199
5.33 Tang::ComputedExpressionFloat Class Reference	199
5.33.1 Detailed Description	201
5.33.2 Constructor & Destructor Documentation	201
5.33.2.1 ComputedExpressionFloat()	201
5.33.3 Member Function Documentation	201
5.33.3.1 __add()	201
5.33.3.2 __asCode()	203
5.33.3.3 __assign_index()	203

5.33.3.4	__boolean()	204
5.33.3.5	__divide()	204
5.33.3.6	__equal()	205
5.33.3.7	__float()	205
5.33.3.8	__getIterator()	205
5.33.3.9	__index()	206
5.33.3.10	__integer()	206
5.33.3.11	__iteratorNext()	206
5.33.3.12	__lessThan()	207
5.33.3.13	__modulo()	207
5.33.3.14	__multiply()	208
5.33.3.15	__negative()	208
5.33.3.16	__not()	209
5.33.3.17	__slice()	209
5.33.3.18	__string()	209
5.33.3.19	__subtract()	210
5.33.3.20	dump()	211
5.33.3.21	getValue()	211
5.33.3.22	is_equal() [1/6]	211
5.33.3.23	is_equal() [2/6]	211
5.33.3.24	is_equal() [3/6]	212
5.33.3.25	is_equal() [4/6]	212
5.33.3.26	is_equal() [5/6]	213
5.33.3.27	is_equal() [6/6]	213
5.33.3.28	isCopyNeeded()	213
5.33.3.29	makeCopy()	214
5.34	Tang::ComputedExpressionInteger Class Reference	214
5.34.1	Detailed Description	216
5.34.2	Constructor & Destructor Documentation	216
5.34.2.1	ComputedExpressionInteger()	216
5.34.3	Member Function Documentation	216
5.34.3.1	__add()	216
5.34.3.2	__asCode()	218
5.34.3.3	__assign_index()	218
5.34.3.4	__boolean()	219
5.34.3.5	__divide()	219
5.34.3.6	__equal()	220
5.34.3.7	__float()	220
5.34.3.8	__getIterator()	220
5.34.3.9	__index()	221
5.34.3.10	__integer()	221
5.34.3.11	__iteratorNext()	221

5.34.3.12	<code>__lessThan()</code>	222
5.34.3.13	<code>__modulo()</code>	222
5.34.3.14	<code>__multiply()</code>	223
5.34.3.15	<code>__negative()</code>	224
5.34.3.16	<code>__not()</code>	224
5.34.3.17	<code>__slice()</code>	224
5.34.3.18	<code>__string()</code>	225
5.34.3.19	<code>__subtract()</code>	225
5.34.3.20	<code>dump()</code>	226
5.34.3.21	<code>getValue()</code>	226
5.34.3.22	<code>is_equal()</code> [1/6]	226
5.34.3.23	<code>is_equal()</code> [2/6]	227
5.34.3.24	<code>is_equal()</code> [3/6]	227
5.34.3.25	<code>is_equal()</code> [4/6]	227
5.34.3.26	<code>is_equal()</code> [5/6]	228
5.34.3.27	<code>is_equal()</code> [6/6]	228
5.34.3.28	<code>isCopyNeeded()</code>	229
5.34.3.29	<code>makeCopy()</code>	229
5.35	<code>Tang::ComputedExpressionIterator</code> Class Reference	229
5.35.1	Detailed Description	231
5.35.2	Constructor & Destructor Documentation	231
5.35.2.1	<code>ComputedExpressionIterator()</code>	232
5.35.3	Member Function Documentation	232
5.35.3.1	<code>__add()</code>	232
5.35.3.2	<code>__asCode()</code>	232
5.35.3.3	<code>__assign_index()</code>	233
5.35.3.4	<code>__boolean()</code>	233
5.35.3.5	<code>__divide()</code>	233
5.35.3.6	<code>__equal()</code>	234
5.35.3.7	<code>__float()</code>	234
5.35.3.8	<code>__getIterator()</code>	234
5.35.3.9	<code>__index()</code>	235
5.35.3.10	<code>__integer()</code>	235
5.35.3.11	<code>__iteratorNext()</code>	235
5.35.3.12	<code>__lessThan()</code>	236
5.35.3.13	<code>__modulo()</code>	236
5.35.3.14	<code>__multiply()</code>	237
5.35.3.15	<code>__negative()</code>	237
5.35.3.16	<code>__not()</code>	237
5.35.3.17	<code>__slice()</code>	238
5.35.3.18	<code>__string()</code>	238
5.35.3.19	<code>__subtract()</code>	238

5.35.3.20 dump()	239
5.35.3.21 is_equal() [1/6]	239
5.35.3.22 is_equal() [2/6]	240
5.35.3.23 is_equal() [3/6]	241
5.35.3.24 is_equal() [4/6]	241
5.35.3.25 is_equal() [5/6]	242
5.35.3.26 is_equal() [6/6]	242
5.35.3.27 isCopyNeeded()	242
5.35.3.28 makeCopy()	243
5.36 Tang::ComputedExpressionIteratorEnd Class Reference	243
5.36.1 Detailed Description	245
5.36.2 Member Function Documentation	245
5.36.2.1 __add()	245
5.36.2.2 __asCode()	245
5.36.2.3 __assign_index()	246
5.36.2.4 __boolean()	246
5.36.2.5 __divide()	246
5.36.2.6 __equal()	247
5.36.2.7 __float()	247
5.36.2.8 __getIterator()	247
5.36.2.9 __index()	248
5.36.2.10 __integer()	248
5.36.2.11 __iteratorNext()	248
5.36.2.12 __lessThan()	249
5.36.2.13 __modulo()	249
5.36.2.14 __multiply()	250
5.36.2.15 __negative()	250
5.36.2.16 __not()	250
5.36.2.17 __slice()	250
5.36.2.18 __string()	251
5.36.2.19 __subtract()	251
5.36.2.20 dump()	252
5.36.2.21 is_equal() [1/6]	252
5.36.2.22 is_equal() [2/6]	252
5.36.2.23 is_equal() [3/6]	253
5.36.2.24 is_equal() [4/6]	253
5.36.2.25 is_equal() [5/6]	253
5.36.2.26 is_equal() [6/6]	254
5.36.2.27 isCopyNeeded()	254
5.36.2.28 makeCopy()	255
5.37 Tang::ComputedExpressionMap Class Reference	255
5.37.1 Detailed Description	257

5.37.2 Constructor & Destructor Documentation	257
5.37.2.1 ComputedExpressionMap()	258
5.37.3 Member Function Documentation	258
5.37.3.1 __add()	258
5.37.3.2 __asCode()	258
5.37.3.3 __assign_index()	259
5.37.3.4 __boolean()	259
5.37.3.5 __divide()	259
5.37.3.6 __equal()	260
5.37.3.7 __float()	260
5.37.3.8 __getIterator()	260
5.37.3.9 __index()	261
5.37.3.10 __integer()	261
5.37.3.11 __iteratorNext()	262
5.37.3.12 __lessThan()	262
5.37.3.13 __modulo()	262
5.37.3.14 __multiply()	263
5.37.3.15 __negative()	263
5.37.3.16 __not()	263
5.37.3.17 __slice()	264
5.37.3.18 __string()	264
5.37.3.19 __subtract()	264
5.37.3.20 dump()	265
5.37.3.21 is_equal() [1/6]	265
5.37.3.22 is_equal() [2/6]	266
5.37.3.23 is_equal() [3/6]	267
5.37.3.24 is_equal() [4/6]	267
5.37.3.25 is_equal() [5/6]	268
5.37.3.26 is_equal() [6/6]	268
5.37.3.27 isCopyNeeded()	268
5.37.3.28 makeCopy()	269
5.38 Tang::ComputedExpressionString Class Reference	269
5.38.1 Detailed Description	271
5.38.2 Constructor & Destructor Documentation	271
5.38.2.1 ComputedExpressionString()	271
5.38.3 Member Function Documentation	272
5.38.3.1 __add()	272
5.38.3.2 __asCode()	272
5.38.3.3 __assign_index()	273
5.38.3.4 __boolean()	273
5.38.3.5 __divide()	274
5.38.3.6 __equal()	274

5.38.3.7	<code>__float()</code>	275
5.38.3.8	<code>__getIterator()</code>	275
5.38.3.9	<code>__index()</code>	276
5.38.3.10	<code>__integer()</code>	276
5.38.3.11	<code>__iteratorNext()</code>	276
5.38.3.12	<code>__lessThan()</code>	277
5.38.3.13	<code>__modulo()</code>	278
5.38.3.14	<code>__multiply()</code>	278
5.38.3.15	<code>__negative()</code>	278
5.38.3.16	<code>__not()</code>	279
5.38.3.17	<code>__slice()</code>	279
5.38.3.18	<code>__string()</code>	280
5.38.3.19	<code>__subtract()</code>	280
5.38.3.20	<code>dump()</code>	281
5.38.3.21	<code>is_equal()</code> [1/6]	281
5.38.3.22	<code>is_equal()</code> [2/6]	281
5.38.3.23	<code>is_equal()</code> [3/6]	282
5.38.3.24	<code>is_equal()</code> [4/6]	282
5.38.3.25	<code>is_equal()</code> [5/6]	283
5.38.3.26	<code>is_equal()</code> [6/6]	283
5.38.3.27	<code>isCopyNeeded()</code>	283
5.38.3.28	<code>makeCopy()</code>	284
5.39	Tang::Error Class Reference	284
5.39.1	Detailed Description	285
5.39.2	Constructor & Destructor Documentation	285
5.39.2.1	<code>Error()</code> [1/2]	285
5.39.2.2	<code>Error()</code> [2/2]	285
5.39.3	Friends And Related Function Documentation	286
5.39.3.1	<code>operator<<</code>	286
5.40	Tang::GarbageCollected Class Reference	286
5.40.1	Detailed Description	289
5.40.2	Constructor & Destructor Documentation	289
5.40.2.1	<code>GarbageCollected()</code> [1/3]	289
5.40.2.2	<code>GarbageCollected()</code> [2/3]	289
5.40.2.3	<code>~GarbageCollected()</code>	289
5.40.2.4	<code>GarbageCollected()</code> [3/3]	290
5.40.3	Member Function Documentation	290
5.40.3.1	<code>isCopyNeeded()</code>	290
5.40.3.2	<code>make()</code>	290
5.40.3.3	<code>makeCopy()</code>	291
5.40.3.4	<code>operator"!()</code>	291
5.40.3.5	<code>operator"!=(</code>	292

5.40.3.6 operator%()	292
5.40.3.7 operator*() [1/2]	293
5.40.3.8 operator*() [2/2]	293
5.40.3.9 operator+()	294
5.40.3.10 operator-() [1/2]	294
5.40.3.11 operator-() [2/2]	295
5.40.3.12 operator->()	295
5.40.3.13 operator/()	296
5.40.3.14 operator<()	296
5.40.3.15 operator<=()	297
5.40.3.16 operator=() [1/2]	297
5.40.3.17 operator=() [2/2]	297
5.40.3.18 operator==() [1/8]	299
5.40.3.19 operator==() [2/8]	299
5.40.3.20 operator==() [3/8]	299
5.40.3.21 operator==() [4/8]	300
5.40.3.22 operator==() [5/8]	300
5.40.3.23 operator==() [6/8]	301
5.40.3.24 operator==() [7/8]	301
5.40.3.25 operator==() [8/8]	301
5.40.3.26 operator>()	302
5.40.3.27 operator>=()	302
5.40.4 Friends And Related Function Documentation	303
5.40.4.1 operator<<	303
5.41 Tang::HtmlEscape Class Reference	303
5.41.1 Detailed Description	304
5.41.2 Constructor & Destructor Documentation	304
5.41.2.1 HtmlEscape()	305
5.41.3 Member Function Documentation	305
5.41.3.1 get_next_token()	305
5.42 Tang::HtmlEscapeAscii Class Reference	305
5.42.1 Detailed Description	306
5.42.2 Constructor & Destructor Documentation	306
5.42.2.1 HtmlEscapeAscii()	307
5.42.3 Member Function Documentation	307
5.42.3.1 get_next_token()	307
5.43 Tang::location Class Reference	307
5.43.1 Detailed Description	309
5.44 Tang::position Class Reference	309
5.44.1 Detailed Description	310
5.45 Tang::Program Class Reference	310
5.45.1 Detailed Description	313

5.45.2 Member Enumeration Documentation	313
5.45.2.1 CodeType	313
5.45.3 Constructor & Destructor Documentation	313
5.45.3.1 Program()	313
5.45.4 Member Function Documentation	314
5.45.4.1 addBreak()	314
5.45.4.2 addBytecode()	314
5.45.4.3 addContinue()	315
5.45.4.4 addIdentifier()	315
5.45.4.5 addIdentifierAssigned()	315
5.45.4.6 addString()	315
5.45.4.7 dumpBytecode()	316
5.45.4.8 execute()	316
5.45.4.9 getAst()	316
5.45.4.10 getBytecode()	317
5.45.4.11 getCode()	317
5.45.4.12 getIdentifiers()	317
5.45.4.13 getIdentifiersAssigned()	317
5.45.4.14 getResult()	318
5.45.4.15 getStrings()	318
5.45.4.16 popBreakStack()	318
5.45.4.17 popContinueStack()	319
5.45.4.18 pushEnvironment()	319
5.45.4.19 setFunctionStackDeclaration()	320
5.45.4.20 setJumpTarget()	320
5.45.5 Member Data Documentation	320
5.45.5.1 functionsDeclared	320
5.46 Tang::SingletonObjectPool< T > Class Template Reference	321
5.46.1 Detailed Description	322
5.46.2 Member Function Documentation	322
5.46.2.1 get()	322
5.46.2.2 getInstance()	323
5.46.2.3 recycle()	323
5.46.3 Member Data Documentation	323
5.46.3.1 currentIndex	323
5.46.3.2 currentRecycledIndex	323
5.47 Tang::TangBase Class Reference	324
5.47.1 Detailed Description	324
5.47.2 Constructor & Destructor Documentation	324
5.47.2.1 TangBase()	324
5.47.3 Member Function Documentation	324
5.47.3.1 compileScript()	324

5.48 Tang::TangScanner Class Reference	325
5.48.1 Detailed Description	326
5.48.2 Constructor & Destructor Documentation	327
5.48.2.1 TangScanner()	327
5.48.3 Member Function Documentation	327
5.48.3.1 get_next_token()	327
5.49 Tang::Unescape Class Reference	328
5.49.1 Detailed Description	328
5.49.2 Constructor & Destructor Documentation	329
5.49.2.1 Unescape()	329
5.49.3 Member Function Documentation	329
5.49.3.1 get_next_token()	329
5.50 Tang::UnicodeString Class Reference	330
5.50.1 Detailed Description	331
5.50.2 Constructor & Destructor Documentation	331
5.50.2.1 UnicodeString()	331
5.50.3 Member Function Documentation	331
5.50.3.1 bytesLength()	331
5.50.3.2 length()	332
5.50.3.3 operator std::string()	332
5.50.3.4 operator+()	332
5.50.3.5 operator<()	333
5.50.3.6 operator==(())	333
5.50.3.7 substr()	333
6 File Documentation	335
6.1 build/generated/location.hh File Reference	335
6.1.1 Detailed Description	336
6.1.2 Function Documentation	336
6.1.2.1 operator<<() [1/2]	336
6.1.2.2 operator<<() [2/2]	337
6.2 include/astNode.hpp File Reference	337
6.2.1 Detailed Description	338
6.3 include/astNodeArray.hpp File Reference	338
6.3.1 Detailed Description	339
6.4 include/astNodeAssign.hpp File Reference	339
6.4.1 Detailed Description	340
6.5 include/astNodeBinary.hpp File Reference	340
6.5.1 Detailed Description	341
6.6 include/astNodeBlock.hpp File Reference	341
6.6.1 Detailed Description	342
6.7 include/astNodeBoolean.hpp File Reference	342

6.7.1 Detailed Description	343
6.8 include/astNodeBreak.hpp File Reference	343
6.8.1 Detailed Description	344
6.9 include/astNodeCast.hpp File Reference	344
6.9.1 Detailed Description	345
6.10 include/astNodeContinue.hpp File Reference	345
6.10.1 Detailed Description	346
6.11 include/astNodeDoWhile.hpp File Reference	346
6.11.1 Detailed Description	347
6.12 include/astNodeFloat.hpp File Reference	347
6.12.1 Detailed Description	348
6.13 include/astNodeFor.hpp File Reference	348
6.13.1 Detailed Description	349
6.14 include/astNodeFunctionCall.hpp File Reference	349
6.14.1 Detailed Description	350
6.15 include/astNodeFunctionDeclaration.hpp File Reference	350
6.15.1 Detailed Description	351
6.16 include/astNodeIdentifier.hpp File Reference	351
6.16.1 Detailed Description	352
6.17 include/astNodeIfElse.hpp File Reference	352
6.17.1 Detailed Description	353
6.18 include/astNodeIndex.hpp File Reference	353
6.18.1 Detailed Description	354
6.19 include/astNodeInteger.hpp File Reference	354
6.19.1 Detailed Description	355
6.20 include/astNodeMap.hpp File Reference	355
6.20.1 Detailed Description	356
6.21 include/astNodePrint.hpp File Reference	356
6.21.1 Detailed Description	357
6.22 include/astNodeRangedFor.hpp File Reference	357
6.22.1 Detailed Description	358
6.23 include/astNodeReturn.hpp File Reference	358
6.23.1 Detailed Description	359
6.24 include/astNodeSlice.hpp File Reference	359
6.24.1 Detailed Description	360
6.25 include/astNodeString.hpp File Reference	360
6.25.1 Detailed Description	361
6.26 include/astNodeTernary.hpp File Reference	361
6.26.1 Detailed Description	362
6.27 include/astNodeUnary.hpp File Reference	362
6.27.1 Detailed Description	363
6.28 include/astNodeWhile.hpp File Reference	363

6.28.1 Detailed Description	364
6.29 include/computedExpression.hpp File Reference	364
6.29.1 Detailed Description	364
6.30 include/computedExpressionArray.hpp File Reference	365
6.30.1 Detailed Description	365
6.31 include/computedExpressionBoolean.hpp File Reference	366
6.31.1 Detailed Description	366
6.32 include/computedExpressionCompiledFunction.hpp File Reference	367
6.32.1 Detailed Description	367
6.33 include/computedExpressionError.hpp File Reference	368
6.33.1 Detailed Description	368
6.34 include/computedExpressionFloat.hpp File Reference	369
6.34.1 Detailed Description	369
6.35 include/computedExpressionInteger.hpp File Reference	370
6.35.1 Detailed Description	370
6.36 include/computedExpressionIterator.hpp File Reference	371
6.36.1 Detailed Description	371
6.37 include/computedExpressionIteratorEnd.hpp File Reference	372
6.37.1 Detailed Description	372
6.38 include/computedExpressionMap.hpp File Reference	373
6.38.1 Detailed Description	374
6.39 include/computedExpressionString.hpp File Reference	374
6.39.1 Detailed Description	374
6.40 include/error.hpp File Reference	375
6.40.1 Detailed Description	375
6.41 include/garbageCollected.hpp File Reference	376
6.41.1 Detailed Description	376
6.42 include/htmlEscape.hpp File Reference	377
6.42.1 Detailed Description	377
6.43 include/htmlEscapeAscii.hpp File Reference	378
6.43.1 Detailed Description	378
6.44 include/macros.hpp File Reference	379
6.44.1 Detailed Description	379
6.45 include/opcode.hpp File Reference	379
6.45.1 Detailed Description	380
6.45.2 Enumeration Type Documentation	380
6.45.2.1 Opcode	380
6.46 include/program.hpp File Reference	381
6.46.1 Detailed Description	382
6.47 include/singletonObjectPool.hpp File Reference	382
6.47.1 Detailed Description	383
6.48 include/tang.hpp File Reference	383

6.48.1 Detailed Description	384
6.49 include/tangBase.hpp File Reference	384
6.49.1 Detailed Description	385
6.50 include/tangScanner.hpp File Reference	385
6.50.1 Detailed Description	386
6.51 include/unescape.hpp File Reference	386
6.51.1 Detailed Description	387
6.52 include/unicodeString.hpp File Reference	387
6.52.1 Detailed Description	388
6.52.2 Function Documentation	388
6.52.2.1 htmlEscape()	388
6.52.2.2 htmlEscapeAscii()	389
6.52.2.3 unescape()	389
6.53 src/astNode.cpp File Reference	390
6.53.1 Detailed Description	390
6.54 src/astNodeArray.cpp File Reference	391
6.54.1 Detailed Description	391
6.55 src/astNodeAssign.cpp File Reference	391
6.55.1 Detailed Description	392
6.56 src/astNodeBinary.cpp File Reference	392
6.56.1 Detailed Description	393
6.57 src/astNodeBlock.cpp File Reference	393
6.57.1 Detailed Description	393
6.58 src/astNodeBoolean.cpp File Reference	393
6.58.1 Detailed Description	394
6.59 src/astNodeBreak.cpp File Reference	394
6.59.1 Detailed Description	395
6.60 src/astNodeCast.cpp File Reference	395
6.60.1 Detailed Description	395
6.61 src/astNodeContinue.cpp File Reference	395
6.61.1 Detailed Description	396
6.62 src/astNodeDoWhile.cpp File Reference	396
6.62.1 Detailed Description	397
6.63 src/astNodeFloat.cpp File Reference	397
6.63.1 Detailed Description	398
6.64 src/astNodeFor.cpp File Reference	398
6.64.1 Detailed Description	398
6.65 src/astNodeFunctionCall.cpp File Reference	398
6.65.1 Detailed Description	399
6.66 src/astNodeFunctionDeclaration.cpp File Reference	399
6.66.1 Detailed Description	400
6.67 src/astNodeIdentifier.cpp File Reference	400

6.67.1 Detailed Description	401
6.68 src/astNodeIfElse.cpp File Reference	401
6.68.1 Detailed Description	401
6.69 src/astNodeIndex.cpp File Reference	401
6.69.1 Detailed Description	402
6.70 src/astNodeInteger.cpp File Reference	402
6.70.1 Detailed Description	403
6.71 src/astNodeMap.cpp File Reference	403
6.71.1 Detailed Description	403
6.72 src/astNodePrint.cpp File Reference	403
6.72.1 Detailed Description	404
6.73 src/astNodeRangedFor.cpp File Reference	404
6.73.1 Detailed Description	405
6.74 src/astNodeReturn.cpp File Reference	405
6.74.1 Detailed Description	406
6.75 src/astNodeSlice.cpp File Reference	406
6.75.1 Detailed Description	407
6.76 src/astNodeString.cpp File Reference	407
6.76.1 Detailed Description	408
6.77 src/astNodeTernary.cpp File Reference	408
6.77.1 Detailed Description	409
6.78 src/astNodeUnary.cpp File Reference	409
6.78.1 Detailed Description	409
6.79 src/astNodeWhile.cpp File Reference	409
6.79.1 Detailed Description	410
6.80 src/computedExpression.cpp File Reference	410
6.80.1 Detailed Description	411
6.81 src/computedExpressionArray.cpp File Reference	411
6.81.1 Detailed Description	412
6.82 src/computedExpressionBoolean.cpp File Reference	412
6.82.1 Detailed Description	412
6.83 src/computedExpressionCompiledFunction.cpp File Reference	412
6.83.1 Detailed Description	413
6.84 src/computedExpressionError.cpp File Reference	413
6.84.1 Detailed Description	414
6.85 src/computedExpressionFloat.cpp File Reference	414
6.85.1 Detailed Description	414
6.86 src/computedExpressionInteger.cpp File Reference	414
6.86.1 Detailed Description	415
6.87 src/computedExpressionIterator.cpp File Reference	415
6.87.1 Detailed Description	415
6.88 src/computedExpressionIteratorEnd.cpp File Reference	416

6.88.1 Detailed Description	416
6.89 src/computedExpressionMap.cpp File Reference	416
6.89.1 Detailed Description	417
6.90 src/computedExpressionString.cpp File Reference	417
6.90.1 Detailed Description	417
6.91 src/error.cpp File Reference	417
6.91.1 Detailed Description	418
6.91.2 Function Documentation	418
6.91.2.1 operator<<()	418
6.92 src/program-dumpBytecode.cpp File Reference	419
6.92.1 Detailed Description	419
6.92.2 Macro Definition Documentation	419
6.92.2.1 DUMPPROGRAMCHECK	420
6.93 src/program-execute.cpp File Reference	420
6.93.1 Detailed Description	421
6.93.2 Macro Definition Documentation	421
6.93.2.1 EXECUTEPROGRAMCHECK	421
6.93.2.2 STACKCHECK	421
6.94 src/program.cpp File Reference	421
6.94.1 Detailed Description	422
6.95 src/tangBase.cpp File Reference	422
6.95.1 Detailed Description	423
6.96 src/unicodeString.cpp File Reference	423
6.96.1 Detailed Description	423
6.97 test/test.cpp File Reference	423
6.97.1 Detailed Description	425
6.98 test/testGarbageCollected.cpp File Reference	425
6.98.1 Detailed Description	426
6.99 test/testSingletonObjectPool.cpp File Reference	426
6.99.1 Detailed Description	426
6.100 test/testUnicodeString.cpp File Reference	427
6.100.1 Detailed Description	427

Index	429
--------------	------------

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	33
Tang::AstNodeBoolean	37
Tang::AstNodeBreak	41
Tang::AstNodeCast	45
Tang::AstNodeContinue	50
Tang::AstNodeDoWhile	54
Tang::AstNodeFloat	58
Tang::AstNodeFor	62
Tang::AstNodeFunctionCall	67
Tang::AstNodeFunctionDeclaration	70
Tang::AstNodeIdentifier	74
Tang::AstNodeIfElse	79
Tang::AstNodeIndex	84
Tang::AstNodeInteger	89
Tang::AstNodeMap	93
Tang::AstNodePrint	97
Tang::AstNodeRangedFor	102
Tang::AstNodeReturn	106
Tang::AstNodeSlice	110
Tang::AstNodeString	115
Tang::AstNodeTernary	120
Tang::AstNodeUnary	124
Tang::AstNodeWhile	130
Tang::ComputedExpression	134
Tang::ComputedExpressionArray	147
Tang::ComputedExpressionBoolean	161
Tang::ComputedExpressionCompiledFunction	173
Tang::ComputedExpressionError	186
Tang::ComputedExpressionFloat	199
Tang::ComputedExpressionInteger	214
Tang::ComputedExpressionIterator	229

Tang::ComputedExpressionIteratorEnd	243
Tang::ComputedExpressionMap	255
Tang::ComputedExpressionString	269
Tang::Error	284
Tang::GarbageCollected	286
Tang::location	307
Tang::position	309
Tang::Program	310
Tang::SingletonObjectPool< T >	321
Tang::TangBase	324
TangHtmlEscapeAsciiFlexLexer	
Tang::HtmlEscapeAscii	305
TangHtmlEscapeFlexLexer	
Tang::HtmlEscape	303
TangTangFlexLexer	
Tang::TangScanner	325
TangUnescapeFlexLexer	
Tang::Unescape	328
Tang::UnicodeString	330

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	33
Tang::AstNodeBoolean	An AstNode that represents a boolean literal	37
Tang::AstNodeBreak	An AstNode that represents a <code>break</code> statement	41
Tang::AstNodeCast	An AstNode that represents a typecast of an expression	45
Tang::AstNodeContinue	An AstNode that represents a <code>continue</code> statement	50
Tang::AstNodeDoWhile	An AstNode that represents a <code>do..while</code> statement	54
Tang::AstNodeFloat	An AstNode that represents an float literal	58
Tang::AstNodeFor	An AstNode that represents an <code>if()</code> statement	62
Tang::AstNodeFunctionCall	An AstNode that represents a function call	67
Tang::AstNodeFunctionDeclaration	An AstNode that represents a function declaration	70
Tang::AstNodeIdentifier	An AstNode that represents an identifier	74
Tang::AstNodeIfElse	An AstNode that represents an <code>if..else</code> statement	79
Tang::AstNodeIndex	An AstNode that represents an index into a collection	84
Tang::AstNodeInteger	An AstNode that represents an integer literal	89

Tang::AstNodeMap	
An AstNode that represents a map literal	93
Tang::AstNodePrint	
An AstNode that represents a print typeeration	97
Tang::AstNodeRangedFor	
An AstNode that represents a ranged for() statement	102
Tang::AstNodeReturn	
An AstNode that represents a <code>return</code> statement	106
Tang::AstNodeSlice	
An AstNode that represents a ternary expression	110
Tang::AstNodeString	
An AstNode that represents a string literal	115
Tang::AstNodeTernary	
An AstNode that represents a ternary expression	120
Tang::AstNodeUnary	
An AstNode that represents a unary negation	124
Tang::AstNodeWhile	
An AstNode that represents a while statement	130
Tang::ComputedExpression	
Represents the result of a computation that has been executed	134
Tang::ComputedExpressionArray	
Represents an Array that is the result of a computation	147
Tang::ComputedExpressionBoolean	
Represents an Boolean that is the result of a computation	161
Tang::ComputedExpressionCompiledFunction	
Represents a Compiled Function declared in the script	173
Tang::ComputedExpressionError	
Represents a Runtime Error	186
Tang::ComputedExpressionFloat	
Represents a Float that is the result of a computation	199
Tang::ComputedExpressionInteger	
Represents an Integer that is the result of a computation	214
Tang::ComputedExpressionIterator	
Represents an Iterator that is the result of a computation	229
Tang::ComputedExpressionIteratorEnd	
Represents that a collection has no more values through which to iterate	243
Tang::ComputedExpressionMap	
Represents an Map that is the result of a computation	255
Tang::ComputedExpressionString	
Represents a String that is the result of a computation	269
Tang::Error	
Used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error	284
Tang::GarbageCollected	
A container that acts as a resource-counting garbage collector for the specified type	286
Tang::HtmlEscape	
The Flex lexer class for the main Tang language	303
Tang::HtmlEscapeAscii	
The Flex lexer class for the main Tang language	305
Tang::location	
Two points in a source file	307
Tang::position	
A point in a source file	309
Tang::Program	
Represents a compiled script or template that may be executed	310
Tang::SingletonObjectPool< T >	
A thread-safe, singleton object pool of the designated type	321

Tang::TangBase	
The base class for the Tang programming language	324
Tang::TangScanner	
The Flex lexer class for the main Tang language	325
Tang::Unescape	
The Flex lexer class for the main Tang language	328
Tang::UnicodeString	
Represents a UTF-8 encoded string that is Unicode-aware	330

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	335
include/astNode.hpp	
Declare the <code>Tang::AstNode</code> base class	337
include/astNodeArray.hpp	
Declare the <code>Tang::AstNodeArray</code> class	338
include/astNodeAssign.hpp	
Declare the <code>Tang::AstNodeAssign</code> class	339
include/astNodeBinary.hpp	
Declare the <code>Tang::AstNodeBinary</code> class	340
include/astNodeBlock.hpp	
Declare the <code>Tang::AstNodeBlock</code> class	341
include/astNodeBoolean.hpp	
Declare the <code>Tang::AstNodeBoolean</code> class	342
include/astNodeBreak.hpp	
Declare the <code>Tang::AstNodeBreak</code> class	343
include/astNodeCast.hpp	
Declare the <code>Tang::AstNodeCast</code> class	344
include/astNodeContinue.hpp	
Declare the <code>Tang::AstNodeContinue</code> class	345
include/astNodeDoWhile.hpp	
Declare the <code>Tang::AstNodeDoWhile</code> class	346
include/astNodeFloat.hpp	
Declare the <code>Tang::AstNodeFloat</code> class	347
include/astNodeFor.hpp	
Declare the <code>Tang::AstNodeFor</code> class	348
include/astNodeFunctionCall.hpp	
Declare the <code>Tang::AstNodeFunctionCall</code> class	349
include/astNodeFunctionDeclaration.hpp	
Declare the <code>Tang::AstNodeFunctionDeclaration</code> class	350
include/astNodeIdentifier.hpp	
Declare the <code>Tang::AstNodeIdentifier</code> class	351
include/astNodeIfElse.hpp	
Declare the <code>Tang::AstNodeIfElse</code> class	352
include/astNodeIndex.hpp	
Declare the <code>Tang::AstNodeIndex</code> class	353

include/ astNodeInteger.hpp	Declare the Tang::AstNodeInteger class	354
include/ astNodeMap.hpp	Declare the Tang::AstNodeMap class	355
include/ astNodePrint.hpp	Declare the Tang::AstNodePrint class	356
include/ astNodeRangedFor.hpp	Declare the Tang::AstNodeRangedFor class	357
include/ astNodeReturn.hpp	Declare the Tang::AstNodeReturn class	358
include/ astNodeSlice.hpp	Declare the Tang::AstNodeSlice class	359
include/ astNodeString.hpp	Declare the Tang::AstNodeString class	360
include/ astNodeTernary.hpp	Declare the Tang::AstNodeTernary class	361
include/ astNodeUnary.hpp	Declare the Tang::AstNodeUnary class	362
include/ astNodeWhile.hpp	Declare the Tang::AstNodeWhile class	363
include/ computedExpression.hpp	Declare the Tang::ComputedExpression base class	364
include/ computedExpressionArray.hpp	Declare the Tang::ComputedExpressionArray class	365
include/ computedExpressionBoolean.hpp	Declare the Tang::ComputedExpressionBoolean class	366
include/ computedExpressionCompiledFunction.hpp	Declare the Tang::ComputedExpressionCompiledFunction class	367
include/ computedExpressionError.hpp	Declare the Tang::ComputedExpressionError class	368
include/ computedExpressionFloat.hpp	Declare the Tang::ComputedExpressionFloat class	369
include/ computedExpressionInteger.hpp	Declare the Tang::ComputedExpressionInteger class	370
include/ computedExpressionIterator.hpp	Declare the Tang::ComputedExpressionIterator class	371
include/ computedExpressionIteratorEnd.hpp	Declare the Tang::ComputedExpressionIteratorEnd class	372
include/ computedExpressionMap.hpp	Declare the Tang::ComputedExpressionMap class	373
include/ computedExpressionString.hpp	Declare the Tang::ComputedExpressionString class	374
include/ error.hpp	Declare the Tang::Error class used to describe syntax and runtime errors	375
include/ garbageCollected.hpp	Declare the Tang::GarbageCollected class	376
include/ htmlEscape.hpp	Declare the Tang::HtmlEscape used to tokenize a Tang script	377
include/ htmlEscapeAscii.hpp	Declare the Tang::HtmlEscapeAscii used to tokenize a Tang script	378
include/ macros.hpp	Contains generic macros	379
include/ opcode.hpp	Declare the Opcodes used in the Bytecode representation of a program	379
include/ program.hpp	Declare the Tang::Program class used to compile and execute source code	381
include/ singletonObjectPool.hpp	Declare the Tang::SingletonObjectPool class	382

include/tang.hpp	Header file supplied for use by 3rd party code so that they can easily include all necessary headers	383
include/tangBase.hpp	Declare the Tang::TangBase class used to interact with Tang	384
include/tangScanner.hpp	Declare the Tang::TangScanner used to tokenize a Tang script	385
include/unescape.hpp	Declare the Tang::Unescape used to tokenize a Tang script	386
include/unicodeString.hpp	Contains the code to interface with the ICU library	387
src/astNode.cpp	Define the Tang::AstNode class	390
src/astNodeArray.cpp	Define the Tang::AstNodeArray class	391
src/astNodeAssign.cpp	Define the Tang::AstNodeAssign class	391
src/astNodeBinary.cpp	Define the Tang::AstNodeBinary class	392
src/astNodeBlock.cpp	Define the Tang::AstNodeBlock class	393
src/astNodeBoolean.cpp	Define the Tang::AstNodeBoolean class	393
src/astNodeBreak.cpp	Define the Tang::AstNodeBreak class	394
src/astNodeCast.cpp	Define the Tang::AstNodeCast class	395
src/astNodeContinue.cpp	Define the Tang::AstNodeContinue class	395
src/astNodeDoWhile.cpp	Define the Tang::AstNodeDoWhile class	396
src/astNodeFloat.cpp	Define the Tang::AstNodeFloat class	397
src/astNodeFor.cpp	Define the Tang::AstNodeFor class	398
src/astNodeFunctionCall.cpp	Define the Tang::AstNodeFunctionCall class	398
src/astNodeFunctionDeclaration.cpp	Define the Tang::AstNodeFunctionDeclaration class	399
src/astNodeIdentifier.cpp	Define the Tang::AstNodeIdentifier class	400
src/astNodeIfElse.cpp	Define the Tang::AstNodeIfElse class	401
src/astNodeIndex.cpp	Define the Tang::AstNodeIndex class	401
src/astNodeInteger.cpp	Define the Tang::AstNodeInteger class	402
src/astNodeMap.cpp	Define the Tang::AstNodeMap class	403
src/astNodePrint.cpp	Define the Tang::AstNodePrint class	403
src/astNodeRangedFor.cpp	Define the Tang::AstNodeRangedFor class	404
src/astNodeReturn.cpp	Define the Tang::AstNodeReturn class	405
src/astNodeSlice.cpp	Define the Tang::AstNodeSlice class	406

src/astNodeString.cpp	
Define the Tang::AstNodeString class	407
src/astNodeTernary.cpp	
Define the Tang::AstNodeTernary class	408
src/astNodeUnary.cpp	
Define the Tang::AstNodeUnary class	409
src/astNodeWhile.cpp	
Define the Tang::AstNodeWhile class	409
src/computedExpression.cpp	
Define the Tang::ComputedExpression class	410
src/computedExpressionArray.cpp	
Define the Tang::ComputedExpressionArray class	411
src/computedExpressionBoolean.cpp	
Define the Tang::ComputedExpressionBoolean class	412
src/computedExpressionCompiledFunction.cpp	
Define the Tang::ComputedExpressionCompiledFunction class	412
src/computedExpressionError.cpp	
Define the Tang::ComputedExpressionError class	413
src/computedExpressionFloat.cpp	
Define the Tang::ComputedExpressionFloat class	414
src/computedExpressionInteger.cpp	
Define the Tang::ComputedExpressionInteger class	414
src/computedExpressionIterator.cpp	
Define the Tang::ComputedExpressionIterator class	415
src/computedExpressionIteratorEnd.cpp	
Define the Tang::ComputedExpressionIteratorEnd class	416
src/computedExpressionMap.cpp	
Define the Tang::ComputedExpressionMap class	416
src/computedExpressionString.cpp	
Define the Tang::ComputedExpressionString class	417
src/error.cpp	
Define the Tang::Error class	417
src/program-dumpBytecode.cpp	
Define the Tang::Program::dumpBytecode method	419
src/program-execute.cpp	
Define the Tang::Program::execute method	420
src/program.cpp	
Define the Tang::Program class	421
src/tangBase.cpp	
Define the Tang::TangBase class	422
src/unicodeString.cpp	
Contains the function declarations for the Tang::UnicodeString class and the interface to ICU	423
test/test.cpp	
Test the general language behaviors	423
test/testGarbageCollected.cpp	
Test the generic behavior of the Tang::GarbageCollected class	425
test/testSingletonObjectPool.cpp	
Test the generic behavior of the Tang::SingletonObjectPool class	426
test/testUnicodeString.cpp	
Contains tests for the Tang::UnicodeString class	427

Chapter 5

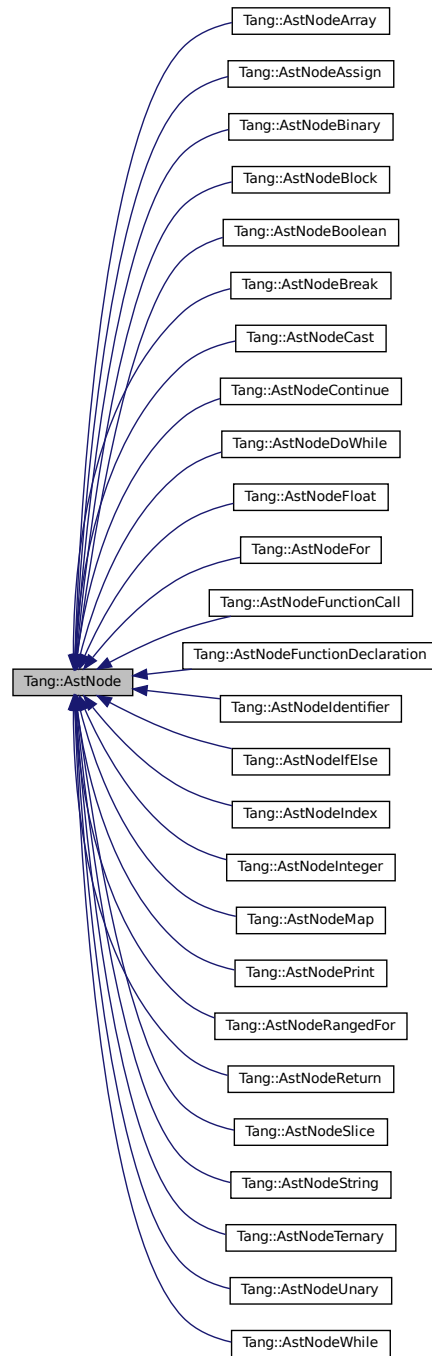
Class Documentation

5.1 Tang::AstNode Class Reference

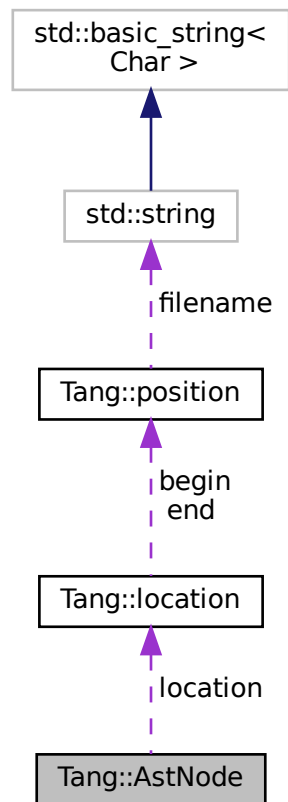
Base class for representing nodes of an Abstract Syntax Tree (AST).

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

Inheritance diagram for Tang::AstNode:



Collaboration diagram for Tang::AstNode:



Public Types

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

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

Public Member Functions

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

Protected Attributes

- [Tang::location location](#)

The location associated with this node.

5.1.1 Detailed Description

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

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

5.1.2 Member Enumeration Documentation

5.1.2.1 PreprocessState

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

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

Enumerator

Default	The default state.
IsAssignment	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::AstNodeMap](#), [Tang::AstNodeInteger](#), [Tang::AstNodeIndex](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFunctionDeclaration](#), [Tang::AstNodeFunctionCall](#), [Tang::AstNodeFor](#), [Tang::AstNodeFloat](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeContinue](#), [Tang::AstNodeCast](#), [Tang::AstNodeBreak](#), [Tang::AstNodeBoolean](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), [Tang::AstNodeAssign](#), and [Tang::AstNodeArray](#).

Here is the call graph for this function:



5.1.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

[Tang::AstNodeIndex](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFunctionDeclaration](#), [Tang::AstNodeFunctionCall](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeCast](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), [Tang::AstNodeAssign](#), and [Tang::AstNodeArray](#).

5.1.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

<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::AstNodeMap](#), [Tang::AstNodeInteger](#), [Tang::AstNodeIndex](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFunctionDeclaration](#), [Tang::AstNodeFunctionCall](#), [Tang::AstNodeFor](#), [Tang::AstNodeFloat](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeContinue](#), [Tang::AstNodeCast](#), [Tang::AstNodeBreak](#), [Tang::AstNodeBoolean](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), [Tang::AstNodeAssign](#), and [Tang::AstNodeArray](#).

Here is the call graph for this function:



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

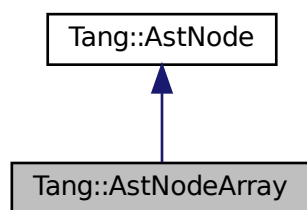
- [include/astNode.hpp](#)
- [src/astNode.cpp](#)

5.2 Tang::AstNodeArray Class Reference

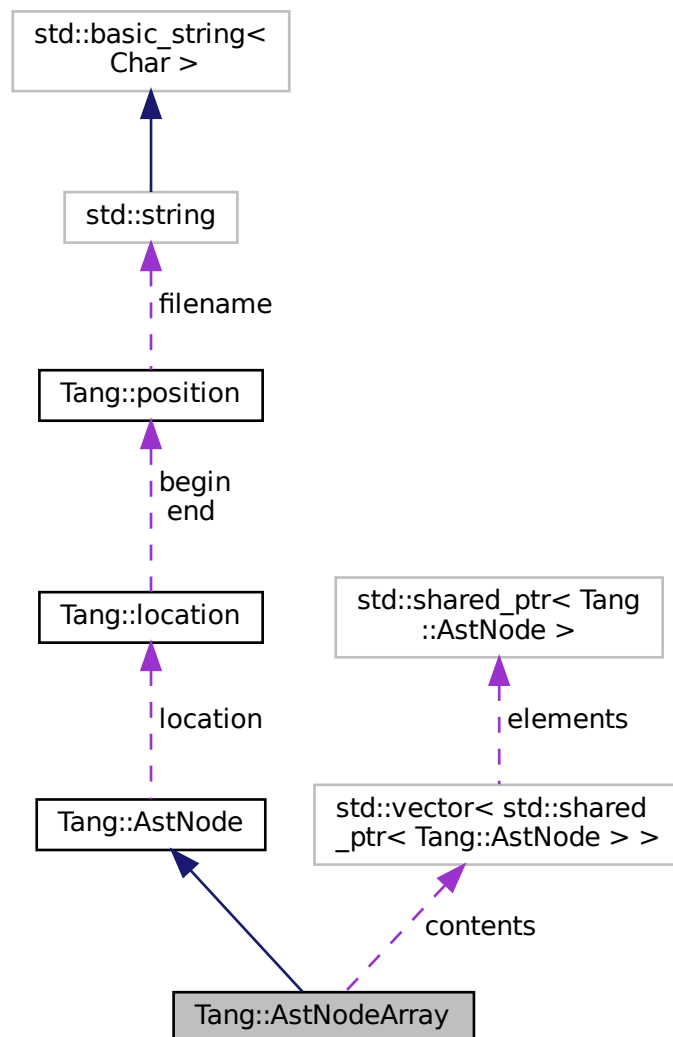
An [AstNode](#) that represents an array literal.

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

Inheritance diagram for Tang::AstNodeArray:



Collaboration diagram for Tang::AstNodeArray:



Public Types

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

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

Public Member Functions

- [AstNodeArray](#) (std::vector< std::shared_ptr< [Tang::AstNode](#) >> [contents](#), [Tang::location](#) [location](#))

The constructor.

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

Return a string that describes the contents of the node.

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

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

- virtual void [compilePreprocess](#) ([Program](#) &program, [PreprocessState](#) state) const override
Run any preprocess analysis needed before compilation.

Protected Attributes

- [Tang::location](#) location
The location associated with this node.

Private Attributes

- std::vector< std::shared_ptr< [Tang::AstNode](#) > > [contents](#)
The contents of the array.

5.2.1 Detailed Description

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

5.2.2 Member Enumeration Documentation

5.2.2.1 PreprocessState

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

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

Enumerator

Default	The default state.
IsAssignment	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]
```

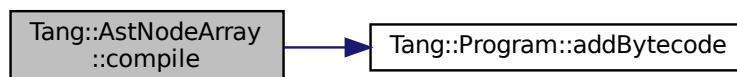
Compile the ast of the provided [Tang::Program](#).

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:

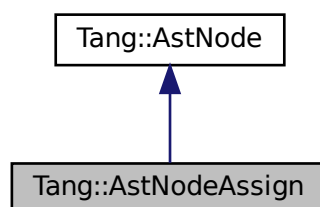
- include/[astNodeArray.hpp](#)
- src/[astNodeArray.cpp](#)

5.3 Tang::AstNodeAssign Class Reference

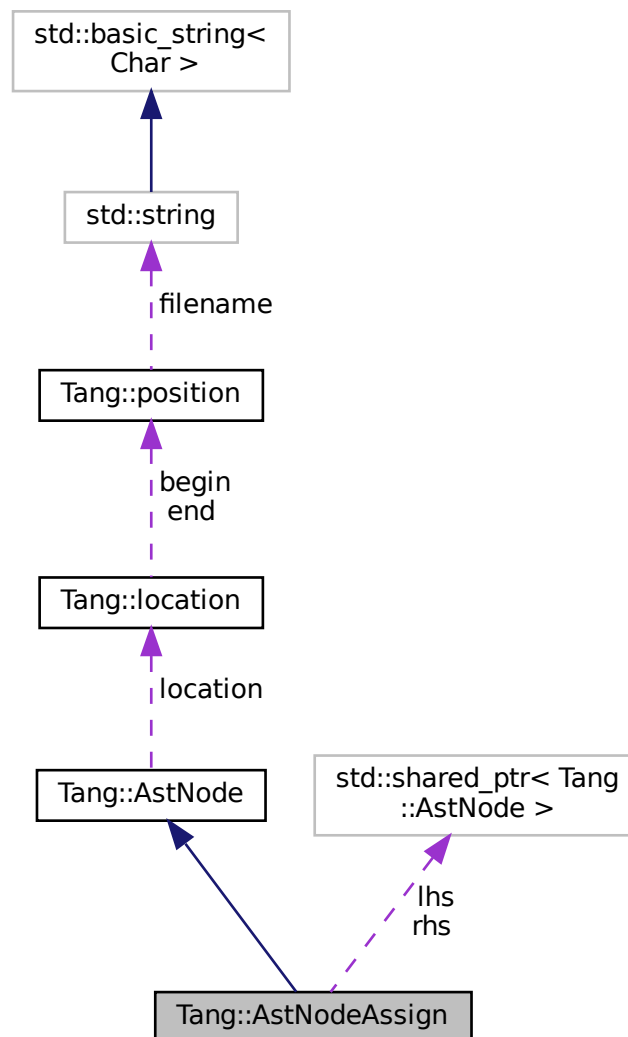
An [AstNode](#) that represents a binary expression.

```
#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.

Protected Attributes

- [Tang::location](#) location

The location associated with this node.

Private Attributes

- std::shared_ptr< [AstNode](#) > lhs

The left hand side expression.

- std::shared_ptr< [AstNode](#) > rhs

The right hand side expression.

5.3.1 Detailed Description

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

5.3.2 Member Enumeration Documentation

5.3.2.1 PreprocessState

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

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

Enumerator

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

5.3.3 Constructor & Destructor Documentation

5.3.3.1 AstNodeAssign()

```
AstNodeAssign::AstNodeAssign (
    std::shared_ptr< AstNode > lhs,
```

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

The constructor.

Parameters

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

5.3.4 Member Function Documentation

5.3.4.1 compile()

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

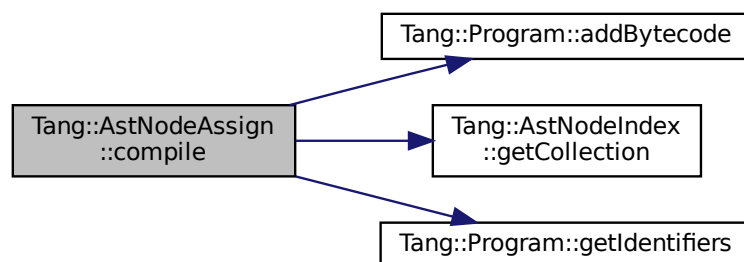
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.3.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.3.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

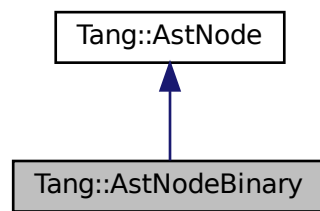
- [include/astNodeAssign.hpp](#)
- [src/astNodeAssign.cpp](#)

5.4 Tang::AstNodeBinary Class Reference

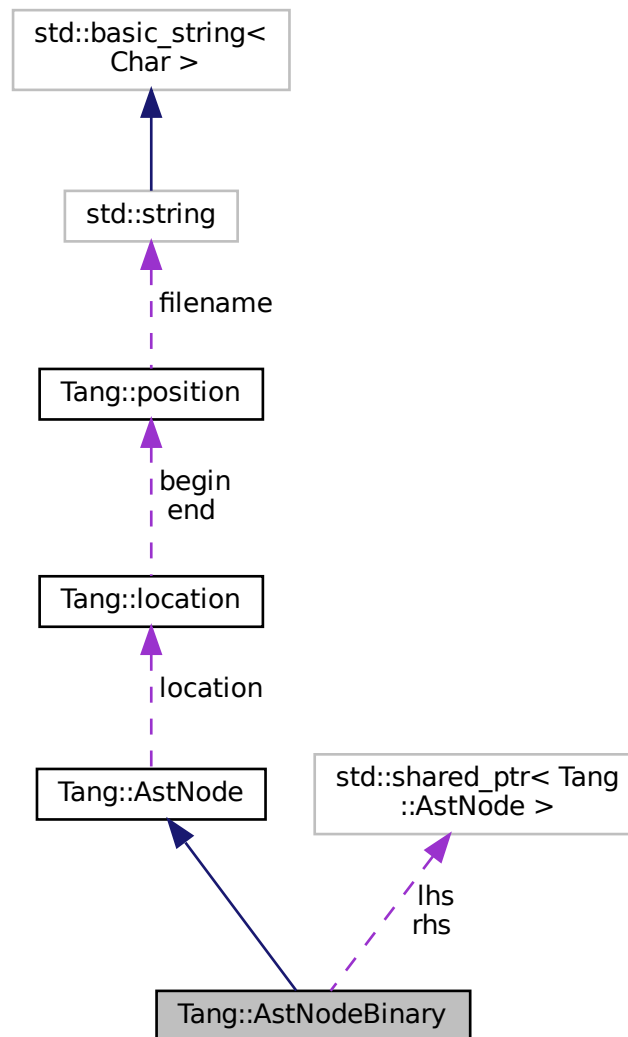
An [AstNode](#) that represents a binary expression.

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

Inheritance diagram for Tang::AstNodeBinary:



Collaboration diagram for Tang::AstNodeBinary:



Public Types

- enum [Operation](#) {
[Add](#) , [Subtract](#) , [Multiply](#) , [Divide](#) ,
[Modulo](#) , [LessThan](#) , [LessThanEqual](#) , [GreaterThan](#) ,
[GreaterThanEqual](#) , [Equal](#) , [NotEqual](#) , [And](#) ,
[Or](#) }

Indicates the type of binary expression that this node represents.

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

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

Public Member Functions

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

Protected Attributes

- [Tang::location](#) location
The location associated with this node.

Private Attributes

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

5.4.1 Detailed Description

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

5.4.2 Member Enumeration Documentation

5.4.2.1 Operation

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

Indicates the type of binary expression that this node represents.

Enumerator

Add	Indicates lhs + rhs.
Subtract	Indicates lhs - rhs.
Multiply	Indicates lhs * rhs.
Divide	Indicates lhs / rhs.
Modulo	Indicates lhs % rhs.
LessThan	Indicates lhs < rhs.
LessThanEqual	Indicates lhs <= rhs.
GreaterThan	Indicates lhs > rhs.
GreaterThanEqual	Indicates lhs >= rhs.
Equal	Indicates lhs == rhs.

5.4.2.2 PreprocessState

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

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

Enumerator

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

5.4.3 Constructor & Destructor Documentation

5.4.3.1 AstNodeBinary()

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

The constructor.

Parameters

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

5.4.4 Member Function Documentation

5.4.4.1 compile()

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

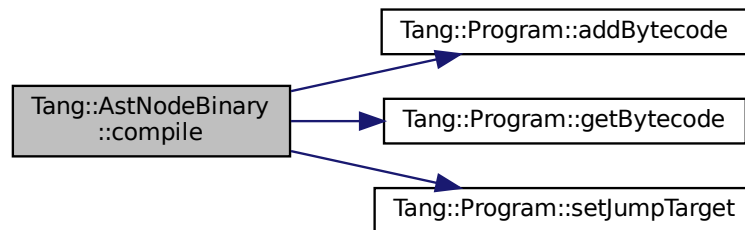
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.4.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.4.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

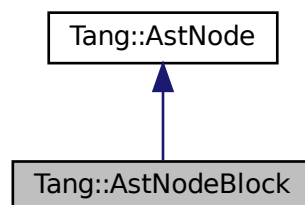
- include/[astNodeBinary.hpp](#)
- src/[astNodeBinary.cpp](#)

5.5 Tang::AstNodeBlock Class Reference

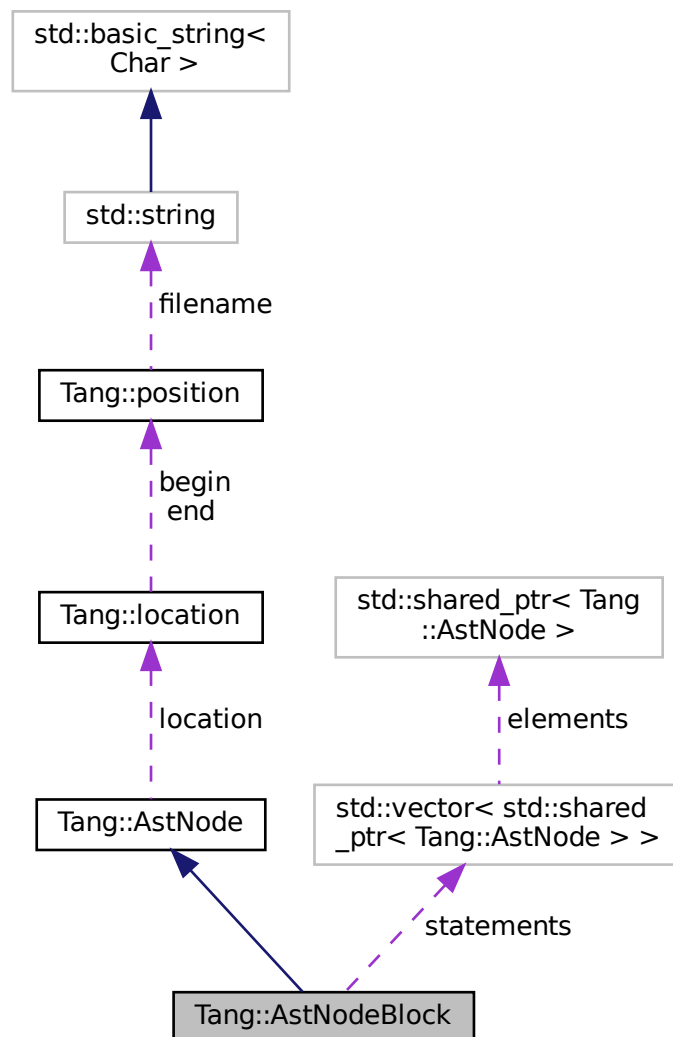
An [AstNode](#) that represents a code block.

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

Inheritance diagram for Tang::AstNodeBlock:



Collaboration diagram for Tang::AstNodeBlock:



Public Types

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

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

Public Member Functions

- [AstNodeBlock](#) (const std::vector< std::shared_ptr< [AstNode](#) > > &statements, [Tang::location](#) location)
The constructor.
- virtual std::string [dump](#) (std::string indent="") const override
Return a string that describes the contents of the node.
- virtual void [compile](#) ([Tang::Program](#) &program) const override

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

- virtual void [compilePreprocess](#) ([Program](#) &program, [PreprocessState](#) state) const override
Run any preprocess analysis needed before compilation.

Protected Attributes

- [Tang::location](#) [location](#)
The location associated with this node.

Private Attributes

- `std::vector< std::shared_ptr< AstNode > > statements`
The statements included in the code block.

5.5.1 Detailed Description

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

5.5.2 Member Enumeration Documentation

5.5.2.1 PreprocessState

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

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

Enumerator

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

5.5.3 Constructor & Destructor Documentation

5.5.3.1 AstNodeBlock()

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

The constructor.

Parameters

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

5.5.4 Member Function Documentation

5.5.4.1 compile()

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

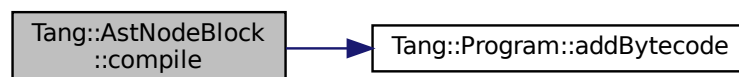
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.5.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.5.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

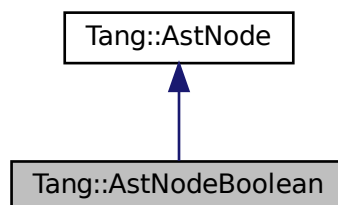
- [include/astNodeBlock.hpp](#)
- [src/astNodeBlock.cpp](#)

5.6 Tang::AstNodeBoolean Class Reference

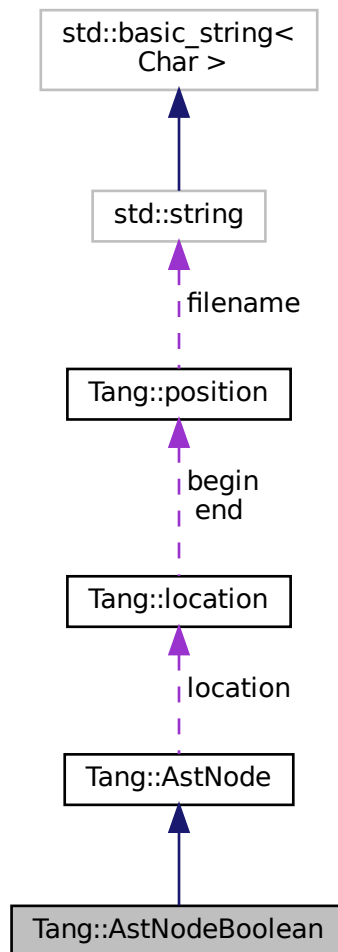
An [AstNode](#) that represents a boolean literal.

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

Inheritance diagram for Tang::AstNodeBoolean:



Collaboration diagram for Tang::AstNodeBoolean:



Public Types

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

Public Member Functions

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

Protected Attributes

- [Tang::location location](#)

The location associated with this node.

Private Attributes

- bool [val](#)

The boolean value being stored.

5.6.1 Detailed Description

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

5.6.2 Member Enumeration Documentation

5.6.2.1 PreprocessState

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

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

Enumerator

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

5.6.3 Constructor & Destructor Documentation

5.6.3.1 AstNodeBoolean()

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

The constructor.

Parameters

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

5.6.4 Member Function Documentation

5.6.4.1 compile()

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

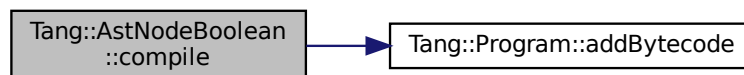
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.6.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.6.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

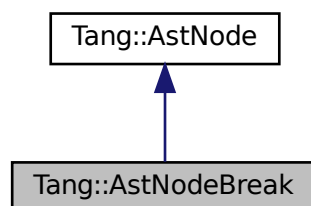
- include/[astNodeBoolean.hpp](#)
- src/[astNodeBoolean.cpp](#)

5.7 Tang::AstNodeBreak Class Reference

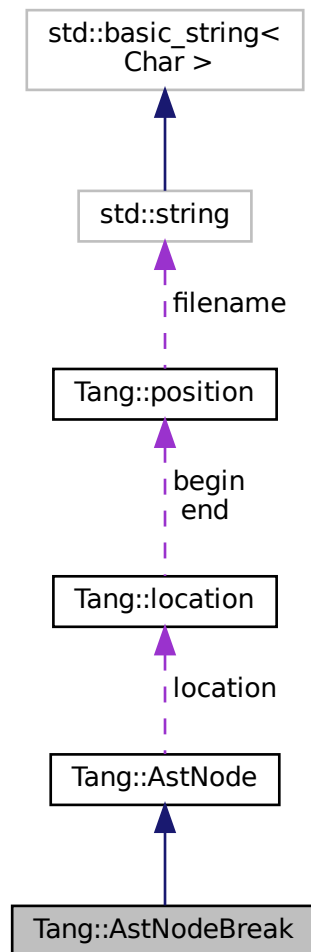
An [AstNode](#) that represents a `break` statement.

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

Inheritance diagram for Tang::AstNodeBreak:



Collaboration diagram for Tang::AstNodeBreak:



Public Types

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

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

Public Member Functions

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

Protected Attributes

- [Tang::location location](#)

The location associated with this node.

5.7.1 Detailed Description

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

5.7.2 Member Enumeration Documentation

5.7.2.1 PreprocessState

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

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

Enumerator

Default	The default state.
IsAssignment	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]
```

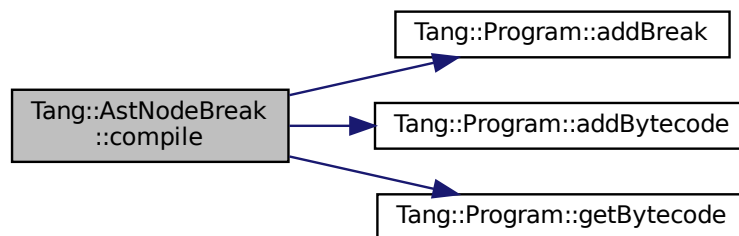
Compile the ast of the provided [Tang::Program](#).

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::AstNodeMap](#), [Tang::AstNodeIndex](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFunctionDeclaration](#), [Tang::AstNodeFunctionCall](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeCast](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), [Tang::AstNodeAssign](#), and [Tang::AstNodeArray](#).

5.7.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

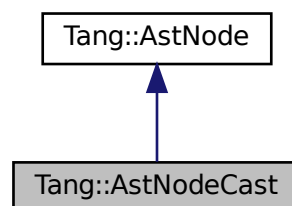
- include/[astNodeBreak.hpp](#)
- src/[astNodeBreak.cpp](#)

5.8 Tang::AstNodeCast Class Reference

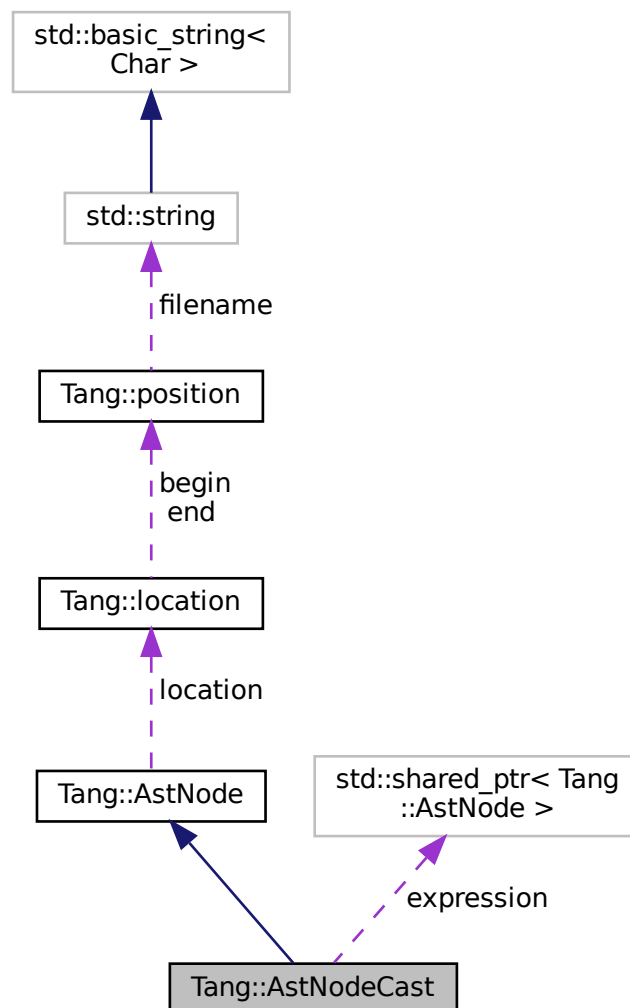
An [AstNode](#) that represents a typecast of an expression.

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

Inheritance diagram for Tang::AstNodeCast:



Collaboration diagram for Tang::AstNodeCast:



Public Types

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

Public Member Functions

- `AstNodeCast` (`Type targetType`, `shared_ptr< AstNode > expression`, `Tang::location location`)
The constructor.
- virtual `std::string dump` (`std::string indent=""`) const override

Return a string that describes the contents of the node.

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

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

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

Run any preprocess analysis needed before compilation.

Protected Attributes

- [Tang::location](#) location

The location associated with this node.

Private Attributes

- [Type](#) targetType

The target type.

- shared_ptr< [AstNode](#) > expression

The expression being typecast.

5.8.1 Detailed Description

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

5.8.2 Member Enumeration Documentation

5.8.2.1 PreprocessState

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

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

Enumerator

Default	The default state.
IsAssignment	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 .
String	Cast to a Tang::ComputedExpressionString .

5.8.3 Constructor & Destructor Documentation

5.8.3.1 AstNodeCast()

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

The constructor.

Parameters

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

5.8.4 Member Function Documentation

5.8.4.1 compile()

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

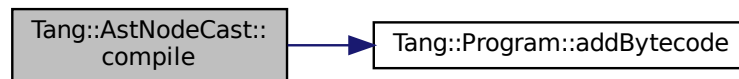
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.8.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.8.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

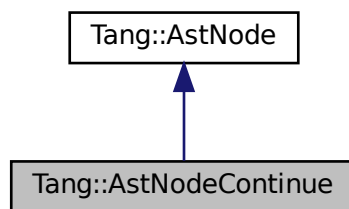
- [include/astNodeCast.hpp](#)
- [src/astNodeCast.cpp](#)

5.9 Tang::AstNodeContinue Class Reference

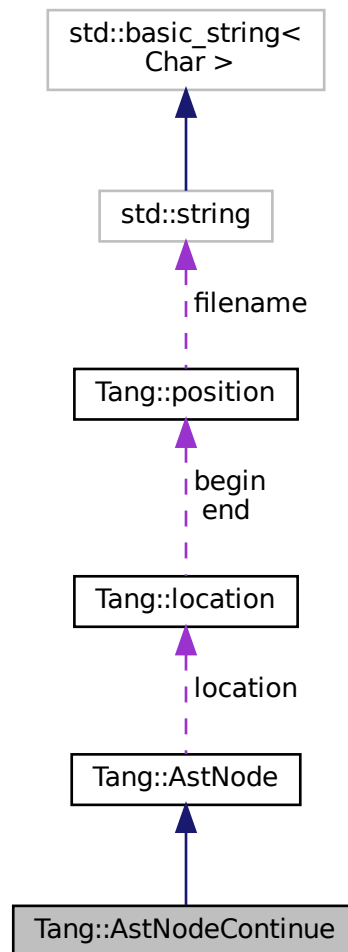
An [AstNode](#) that represents a `continue` statement.

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

Inheritance diagram for Tang::AstNodeContinue:



Collaboration diagram for Tang::AstNodeContinue:



Public Types

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

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

Public Member Functions

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

Protected Attributes

- [Tang::location location](#)

The location associated with this node.

5.9.1 Detailed Description

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

5.9.2 Member Enumeration Documentation

5.9.2.1 PreprocessState

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

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

Enumerator

Default	The default state.
IsAssignment	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]
```

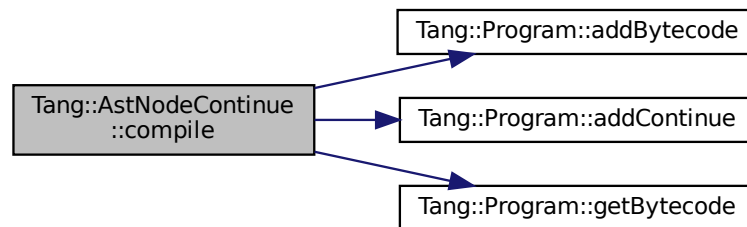
Compile the ast of the provided [Tang::Program](#).

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::AstNodeMap](#), [Tang::AstNodeIndex](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFunctionDeclaration](#), [Tang::AstNodeFunctionCall](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeCast](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), [Tang::AstNodeAssign](#), and [Tang::AstNodeArray](#).

5.9.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

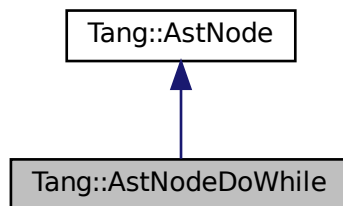
- [include/astNodeContinue.hpp](#)
- [src/astNodeContinue.cpp](#)

5.10 Tang::AstNodeDoWhile Class Reference

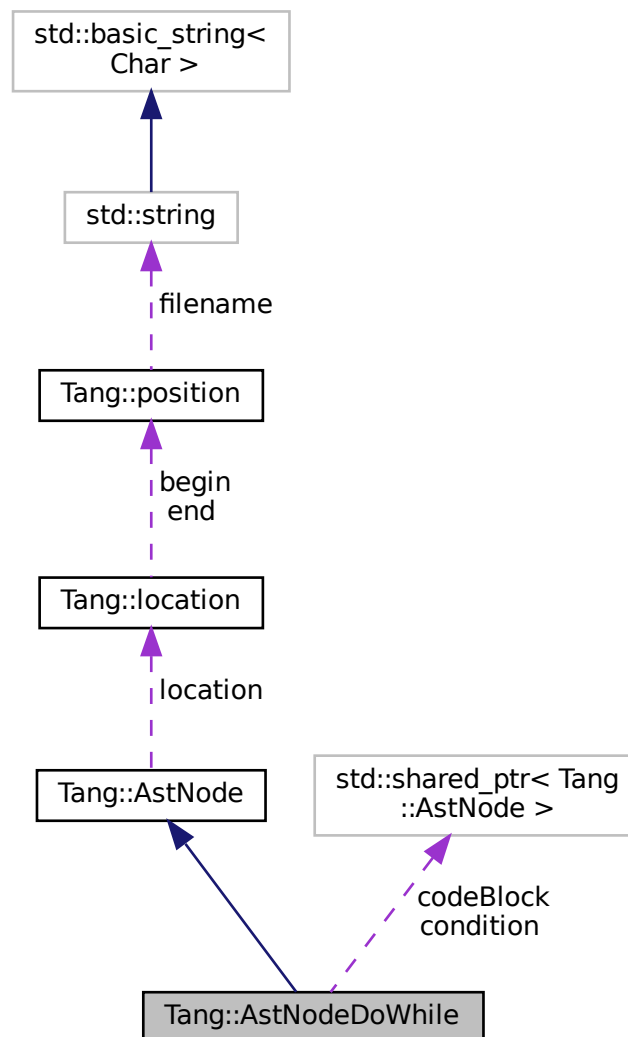
An [AstNode](#) that represents a do..while statement.

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

Inheritance diagram for Tang::AstNodeDoWhile:



Collaboration diagram for Tang::AstNodeDoWhile:



Public Types

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

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

Public Member Functions

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

The constructor.

- virtual std::string `dump` (std::string indent="") const override

Return a string that describes the contents of the node.

- virtual void `compile` ([Tang::Program](#) &program) const override
Compile the ast of the provided [Tang::Program](#).
- virtual void `compilePreprocess` ([Program](#) &program, [PreprocessState](#) state) const override
Run any preprocess analysis needed before compilation.

Protected Attributes

- [Tang::location](#) location
The location associated with this node.

Private Attributes

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

5.10.1 Detailed Description

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

5.10.2 Member Enumeration Documentation

5.10.2.1 PreprocessState

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

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

Enumerator

Default	The default state.
IsAssignment	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]
```

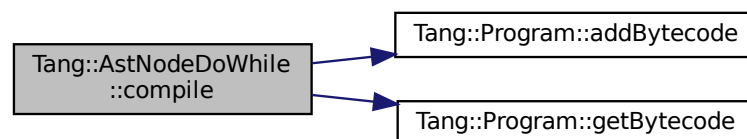
Compile the ast of the provided [Tang::Program](#).

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:

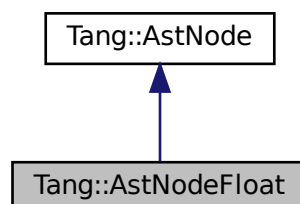
- [include/astNodeDoWhile.hpp](#)
- [src/astNodeDoWhile.cpp](#)

5.11 Tang::AstNodeFloat Class Reference

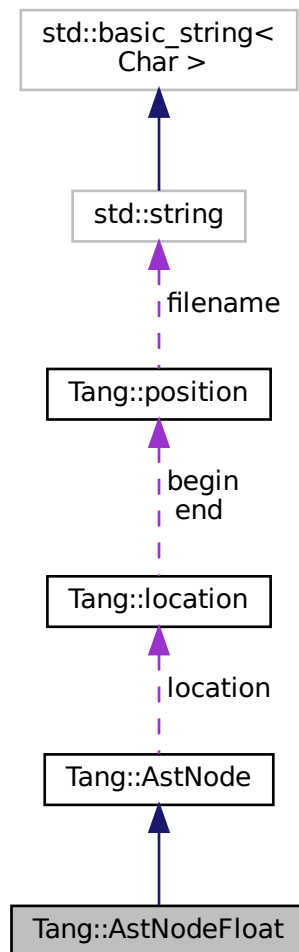
An [AstNode](#) that represents an float literal.

```
#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.

Protected Attributes

- [Tang::location location](#)

The location associated with this node.

Private Attributes

- [Tang::float_t val](#)

The float value being stored.

5.11.1 Detailed Description

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

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

5.11.2 Member Enumeration Documentation

5.11.2.1 PreprocessState

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

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

Enumerator

Default	The default state.
IsAssignment	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]
```

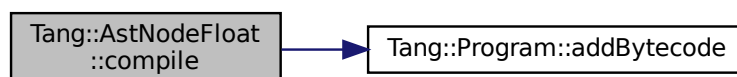
Compile the ast of the provided [Tang::Program](#).

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::AstNodeMap](#), [Tang::AstNodeIndex](#), [Tang::AstNodeIfElse](#), [Tang::AstNodeIdentifier](#), [Tang::AstNodeFunctionDeclaration](#), [Tang::AstNodeFunctionCall](#), [Tang::AstNodeFor](#), [Tang::AstNodeDoWhile](#), [Tang::AstNodeCast](#), [Tang::AstNodeBlock](#), [Tang::AstNodeBinary](#), [Tang::AstNodeAssign](#), and [Tang::AstNodeArray](#).

5.11.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

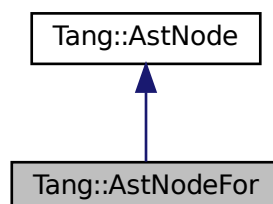
- [include/astNodeFloat.hpp](#)
- [src/astNodeFloat.cpp](#)

5.12 Tang::AstNodeFor Class Reference

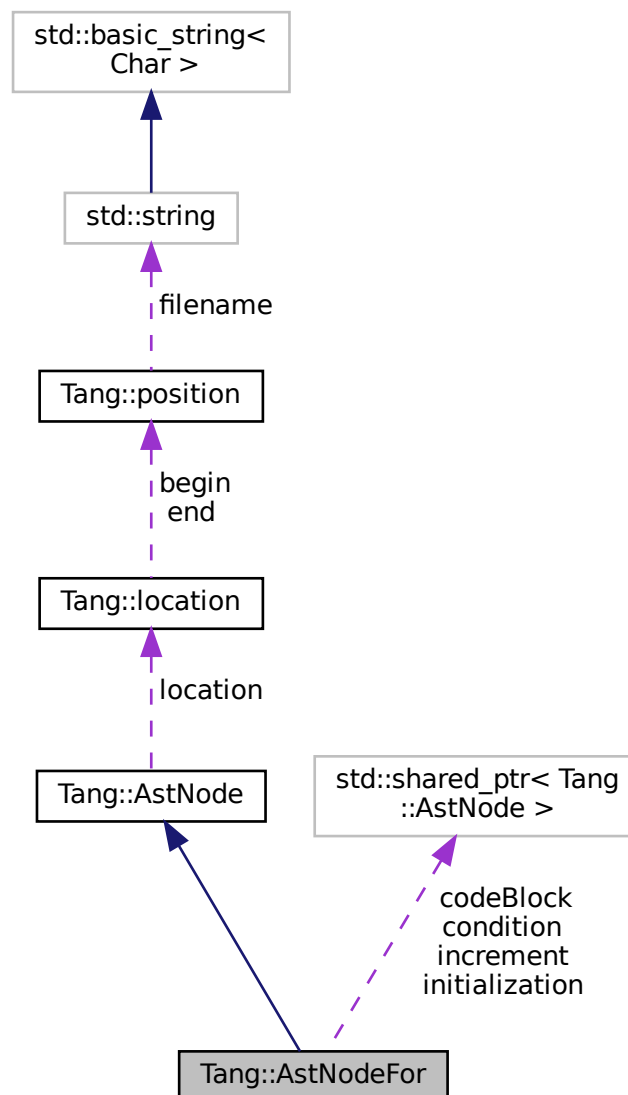
An [AstNode](#) that represents an if() statement.

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

Inheritance diagram for Tang::AstNodeFor:



Collaboration diagram for Tang::AstNodeFor:



Public Types

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

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

Public Member Functions

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

The constructor.

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

Protected Attributes

- [Tang::location](#) location
The location associated with this node.

Private Attributes

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

5.12.1 Detailed Description

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

5.12.2 Member Enumeration Documentation

5.12.2.1 PreprocessState

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

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

Enumerator

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]
```

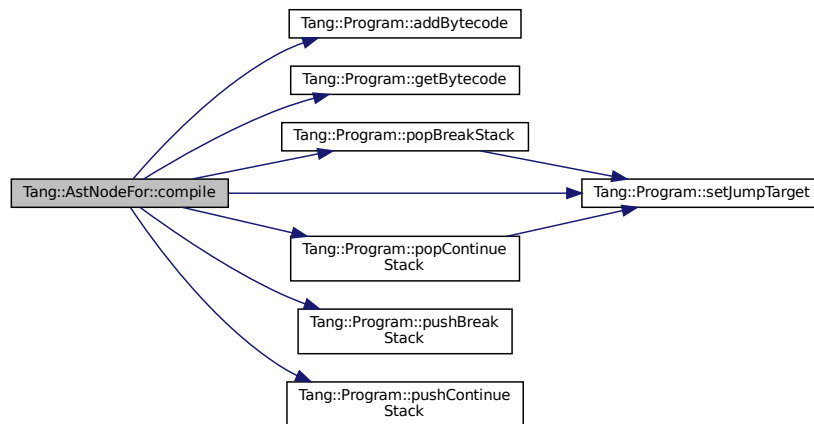
Compile the ast of the provided [Tang::Program](#).

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:

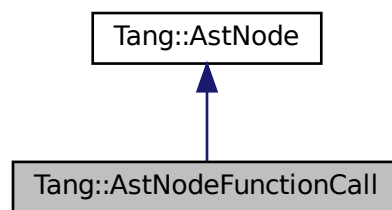
- [include/astNodeFor.hpp](#)
- [src/astNodeFor.cpp](#)

5.13 Tang::AstNodeFunctionCall Class Reference

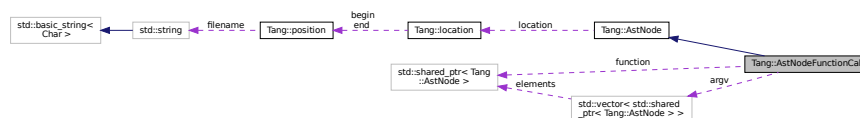
An [AstNode](#) that represents a function call.

```
#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.

Protected Attributes

- [Tang::location location](#)
The location associated with this node.

Private Attributes

- `std::shared_ptr< AstNode > function`
The function being invoked.
- `std::vector< std::shared_ptr< AstNode > > argv`
The list of arguments provided to the function.

5.13.1 Detailed Description

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

5.13.2 Member Enumeration Documentation

5.13.2.1 PreprocessState

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

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

Enumerator

Default	The default state.
IsAssignment	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:

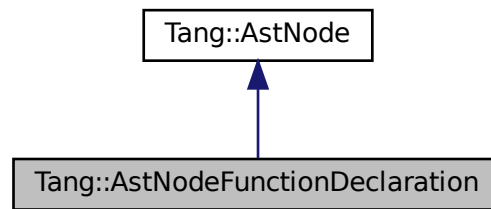
- [include/astNodeFunctionCall.hpp](#)
- [src/astNodeFunctionCall.cpp](#)

5.14 Tang::AstNodeFunctionDeclaration Class Reference

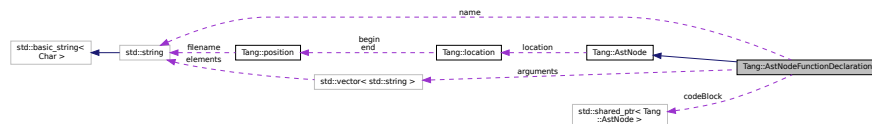
An [AstNode](#) that represents a function declaration.

```
#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.

Protected Attributes

- [Tang::location](#) [location](#)
The location associated with this node.

Private Attributes

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

5.14.1 Detailed Description

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

5.14.2 Member Enumeration Documentation

5.14.2.1 PreprocessState

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

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

Enumerator

Default	The default state.
IsAssignment	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]
```

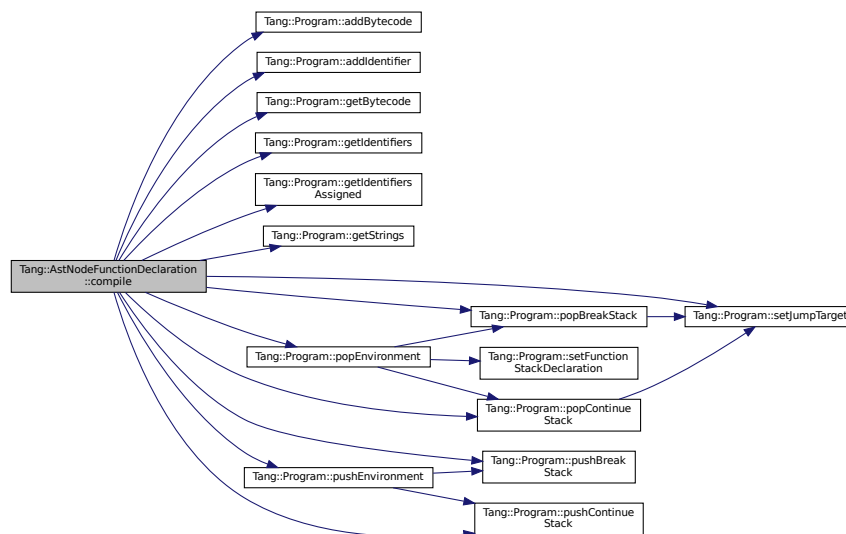
Compile the ast of the provided [Tang::Program](#).

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:

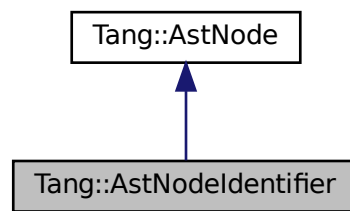
- [include/astNodeFunctionDeclaration.hpp](#)
- [src/astNodeFunctionDeclaration.cpp](#)

5.15 Tang::AstNodeIdentifier Class Reference

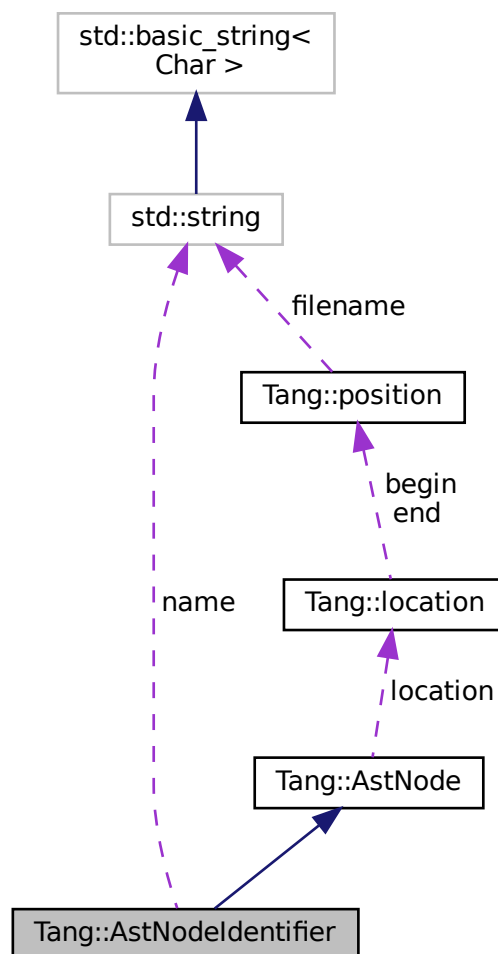
An [AstNode](#) that represents an identifier.

```
#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.

Protected Attributes

- [Tang::location](#) [location](#)
The location associated with this node.

5.15.1 Detailed Description

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

Identifier names are represented by a string.

5.15.2 Member Enumeration Documentation

5.15.2.1 PreprocessState

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

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

Enumerator

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]
```

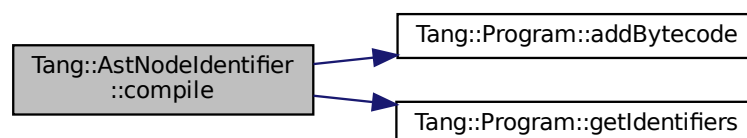
Compile the ast of the provided [Tang::Program](#).

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]
```

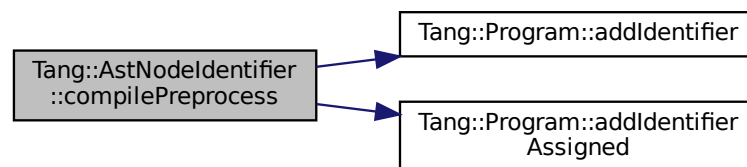
Run any preprocess analysis needed before compilation.

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:

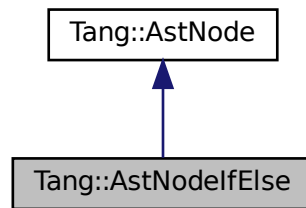
- [include/astNodeIdentifier.hpp](#)
- [src/astNodeIdentifier.cpp](#)

5.16 Tang::AstNodeIfElse Class Reference

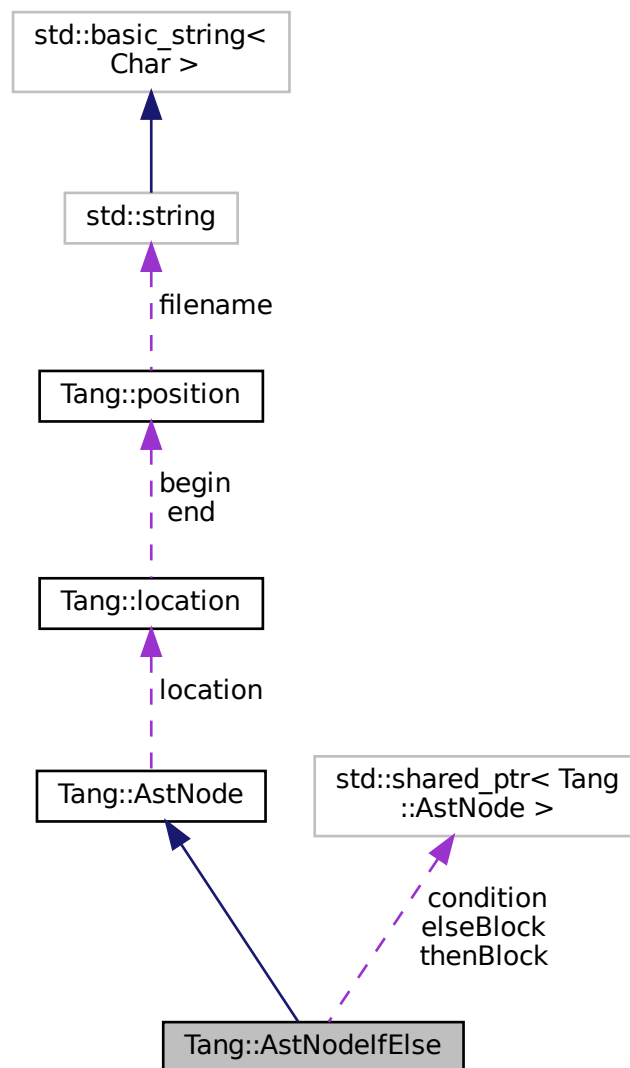
An [AstNode](#) that represents an if..else statement.

```
#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.

Protected Attributes

- [Tang::location location](#)
The location associated with this node.

Private Attributes

- [shared_ptr< AstNode > condition](#)
The expression which determines whether the thenBlock or elseBlock is executed.
- [shared_ptr< AstNode > thenBlock](#)
The statement executed when the condition is true.
- [shared_ptr< AstNode > elseBlock](#)
The statement executed when the condition is false.

5.16.1 Detailed Description

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

5.16.2 Member Enumeration Documentation

5.16.2.1 PreprocessState

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

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

Enumerator

Default	The default state.
IsAssignment	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]
```

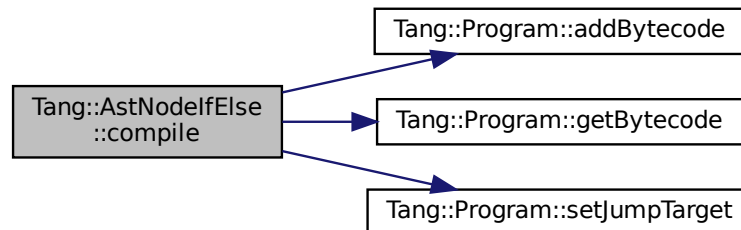
Compile the ast of the provided [Tang::Program](#).

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:

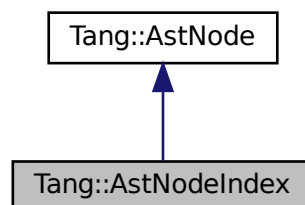
- [include/astNodeIfElse.hpp](#)
- [src/astNodeIfElse.cpp](#)

5.17 Tang::AstNodeIndex Class Reference

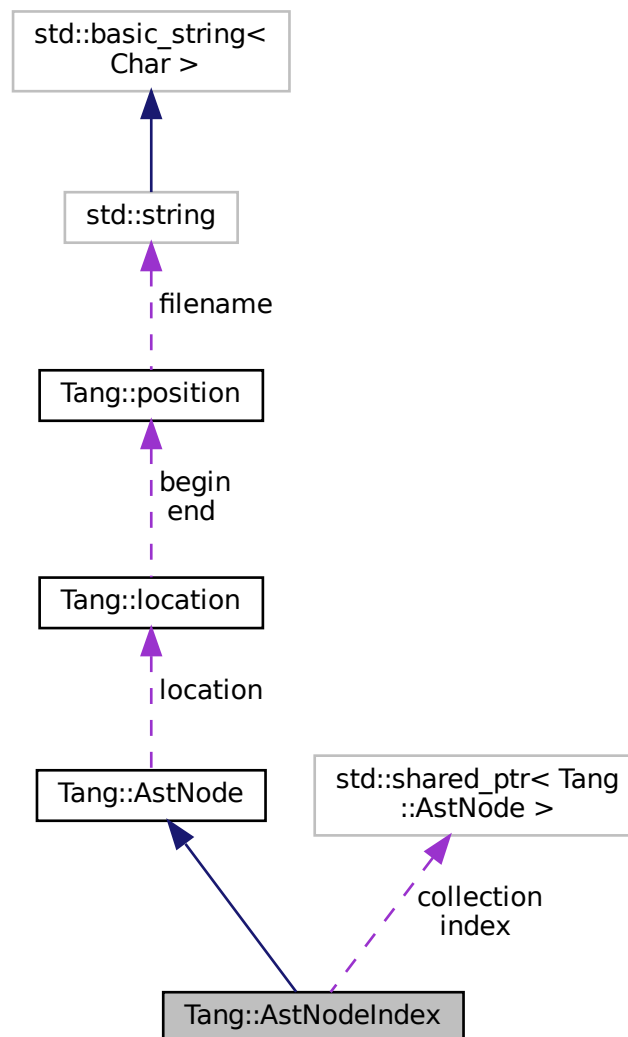
An [AstNode](#) that represents an index into a collection.

```
#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.

Protected Attributes

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

Private Attributes

- std::shared_ptr< `AstNode` > `collection`
The collection into which we will index.
- std::shared_ptr< `AstNode` > `index`
The index expression.

5.17.1 Detailed Description

An `AstNode` that represents an index into a collection.

5.17.2 Member Enumeration Documentation

5.17.2.1 PreprocessState

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

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

Enumerator

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

5.17.3 Constructor & Destructor Documentation

5.17.3.1 AstNodeIndex()

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

The constructor.

Parameters

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

5.17.4 Member Function Documentation

5.17.4.1 compile()

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

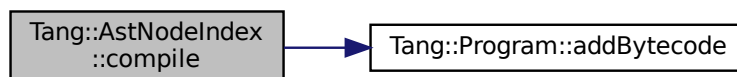
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.17.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.17.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

5.17.4.4 getCollection()

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

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

Returns

The collection into which we will index.

5.17.4.5 getIndex()

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

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

Returns

The index expression.

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

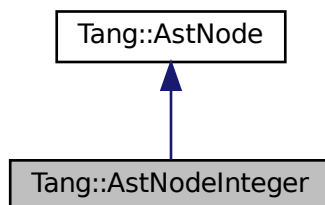
- [include/astNodeIndex.hpp](#)
- [src/astNodeIndex.cpp](#)

5.18 Tang::AstNodeInteger Class Reference

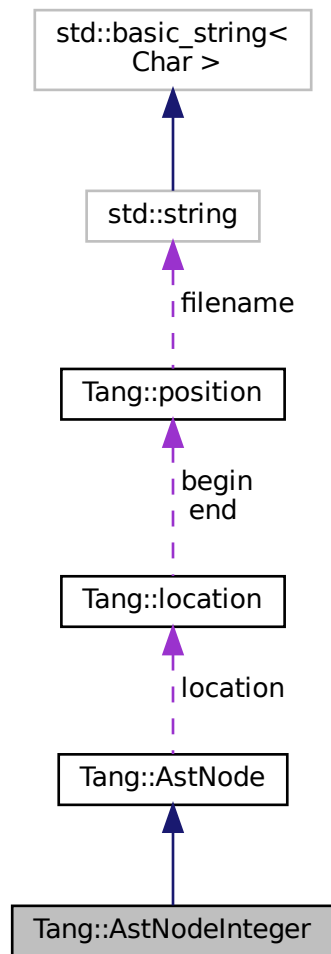
An [AstNode](#) that represents an integer literal.

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

Inheritance diagram for Tang::AstNodeInteger:



Collaboration diagram for Tang::AstNodeInteger:



Public Types

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

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

Public Member Functions

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

Protected Attributes

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

The location associated with this node.

Private Attributes

- [Tang::integer_t](#) *val*

The integer value being stored.

5.18.1 Detailed Description

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

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

5.18.2 Member Enumeration Documentation

5.18.2.1 PreprocessState

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

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

Enumerator

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

5.18.3 Constructor & Destructor Documentation

5.18.3.1 AstNodeInteger()

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

The constructor.

Parameters

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

5.18.4 Member Function Documentation

5.18.4.1 compile()

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

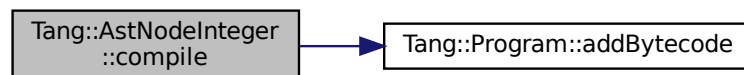
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.18.4.2 compilePreprocess()

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

Run any preprocess analysis needed before compilation.

Parameters

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

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

5.18.4.3 dump()

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

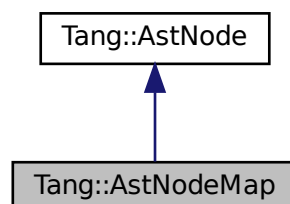
- [include/astNodeInteger.hpp](#)
- [src/astNodeInteger.cpp](#)

5.19 Tang::AstNodeMap Class Reference

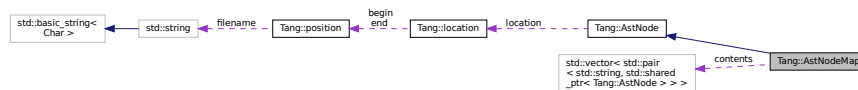
An [AstNode](#) that represents a map literal.

```
#include <astNodeMap.hpp>
```

Inheritance diagram for Tang::AstNodeMap:



Collaboration diagram for Tang::AstNodeMap:



Public Types

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

Public Member Functions

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

Protected Attributes

- [Tang::location](#) location
The location associated with this node.

Private Attributes

- std::vector< std::pair< std::string, std::shared_ptr< [Tang::AstNode](#) > > > [contents](#)
The contents of the array.

5.19.1 Detailed Description

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

Keys can only be strings.

5.19.2 Member Enumeration Documentation

5.19.2.1 PreprocessState

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

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

Enumerator

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

5.19.3 Constructor & Destructor Documentation

5.19.3.1 AstNodeMap()

```
AstNodeMap::AstNodeMap (
    std::vector< std::pair< std::string, std::shared_ptr< Tang::AstNode >>> contents,
    Tang::location location )
```

The constructor.

Parameters

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

5.19.4 Member Function Documentation

5.19.4.1 compile()

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

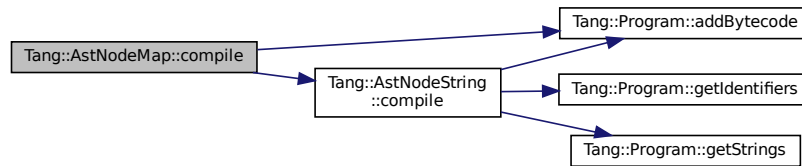
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.19.4.2 compilePreprocess()

```

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

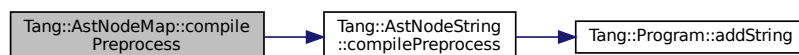
Run any preprocess analysis needed before compilation.

Parameters

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

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

Here is the call graph for this function:



5.19.4.3 dump()

```

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

Return a string that describes the contents of the node.

Parameters

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

Returns

The value as a string.

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

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

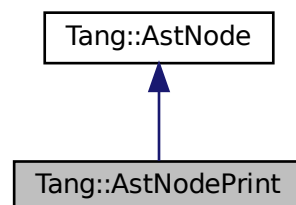
- [include/astNodeMap.hpp](#)
- [src/astNodeMap.cpp](#)

5.20 Tang::AstNodePrint Class Reference

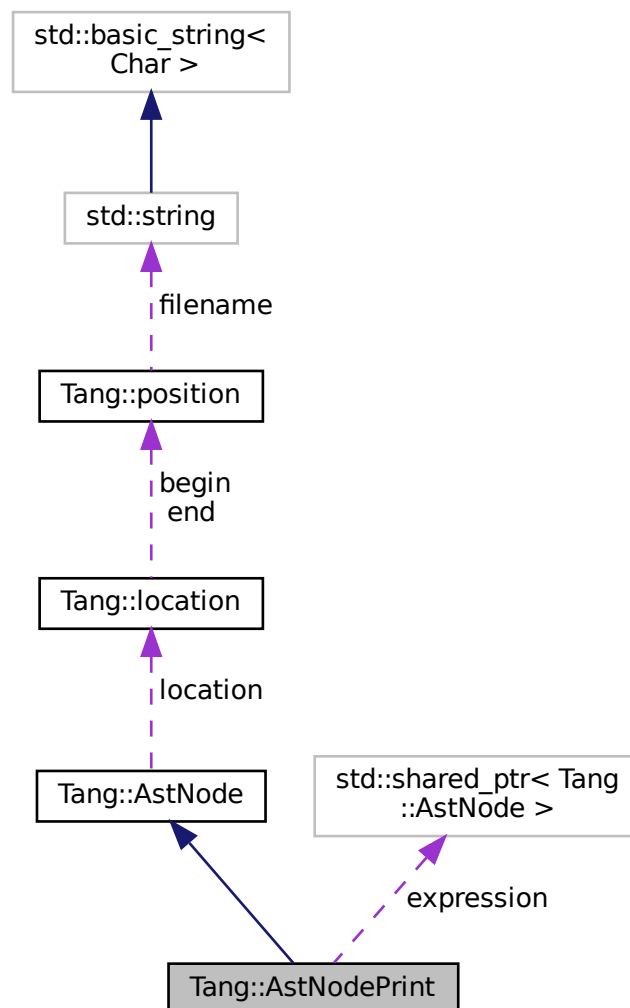
An [AstNode](#) that represents a print typeoperation.

```
#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.

Protected Attributes

- [Tang::location](#) location

The location associated with this node.

Private Attributes

- [Type](#) type

The type of print() being requested.

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

The expression to be printed.

5.20.1 Detailed Description

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

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.

5.20.2.2 Type

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

The type of print() requested.

Enumerator

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

5.20.3 Constructor & Destructor Documentation

5.20.3.1 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.

5.20.4 Member Function Documentation

5.20.4.1 compile()

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

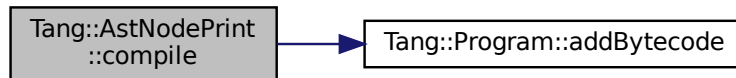
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.20.4.2 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](#).

5.20.4.3 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:

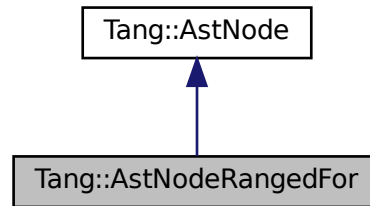
- [include/astNodePrint.hpp](#)
- [src/astNodePrint.cpp](#)

5.21 Tang::AstNodeRangedFor Class Reference

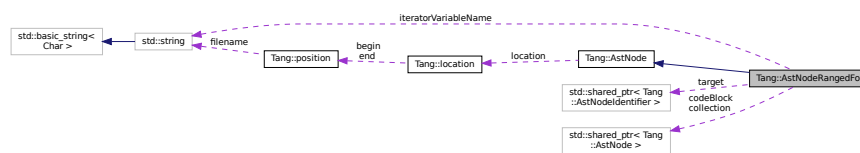
An [AstNode](#) that represents a ranged for() statement.

```
#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.

Protected Attributes

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

The location associated with this node.

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.

5.21.1 Detailed Description

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

5.21.4 Member Function Documentation

5.21.4.1 compile()

```

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

```

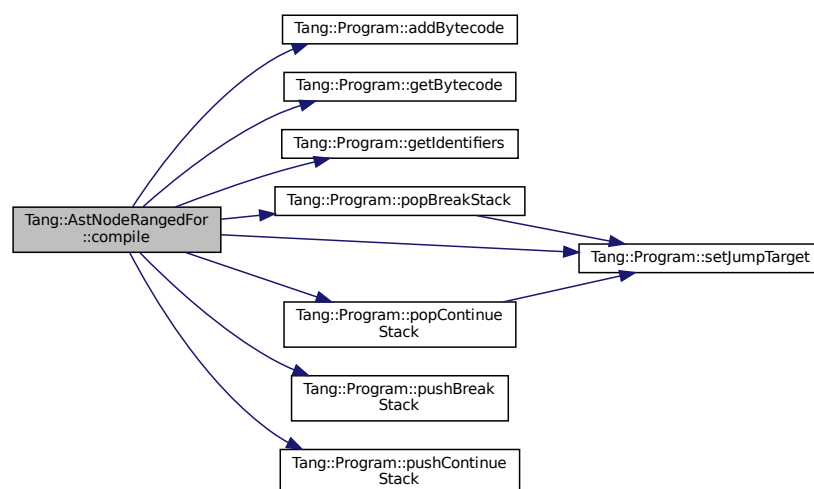
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.21.4.2 compilePreprocess()

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

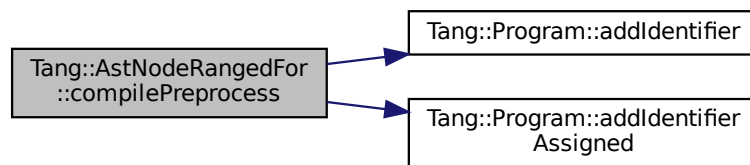
Run any preprocess analysis needed before compilation.

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:



5.21.4.3 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:

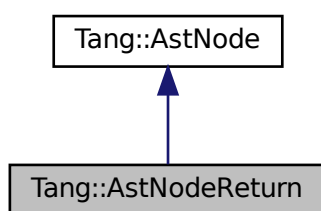
- [include/astNodeRangedFor.hpp](#)
- [src/astNodeRangedFor.cpp](#)

5.22 Tang::AstNodeReturn Class Reference

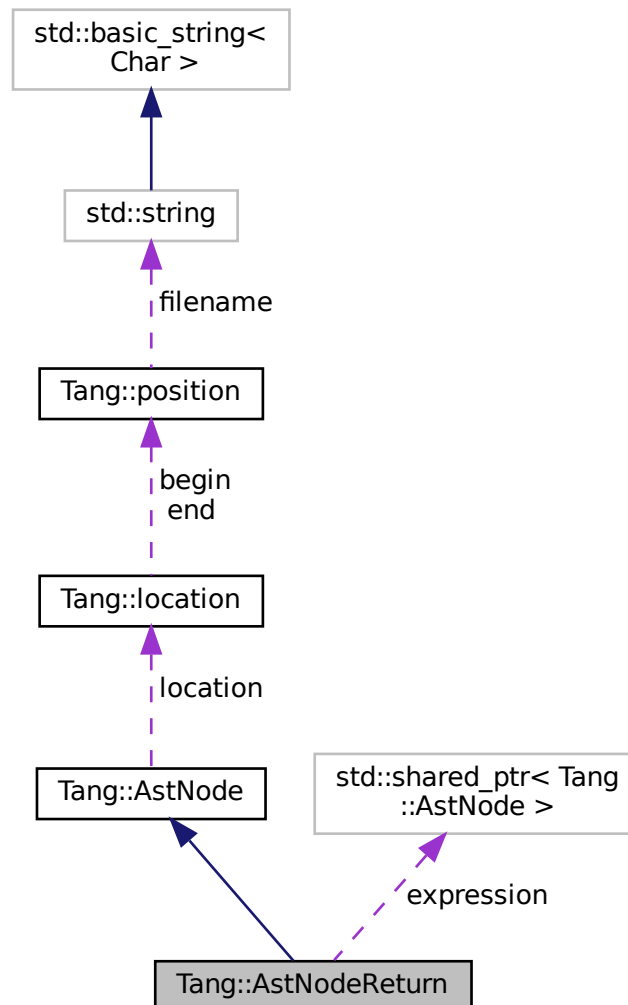
An [AstNode](#) that represents a `return` statement.

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

Inheritance diagram for Tang::AstNodeReturn:



Collaboration diagram for Tang::AstNodeReturn:



Public Types

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

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

Public Member Functions

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

Protected Attributes

- [Tang::location location](#)

The location associated with this node.

Private Attributes

- `shared_ptr< AstNode > expression`

The expression to which the operation will be applied.

5.22.1 Detailed Description

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

5.22.2 Member Enumeration Documentation

5.22.2.1 PreprocessState

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

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

Enumerator

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

5.22.3 Constructor & Destructor Documentation

5.22.3.1 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.

5.22.4 Member Function Documentation

5.22.4.1 compile()

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

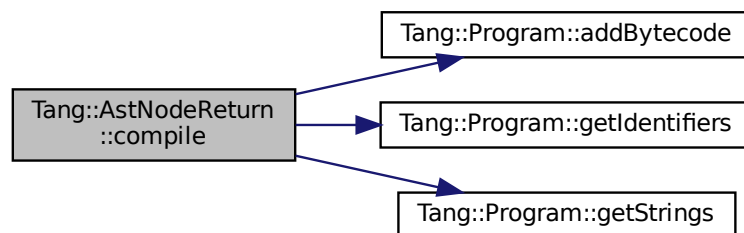
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.22.4.2 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](#).

5.22.4.3 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:

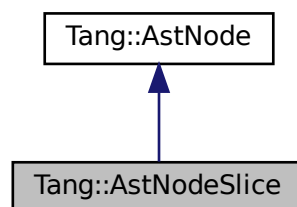
- [include/astNodeReturn.hpp](#)
- [src/astNodeReturn.cpp](#)

5.23 Tang::AstNodeSlice Class Reference

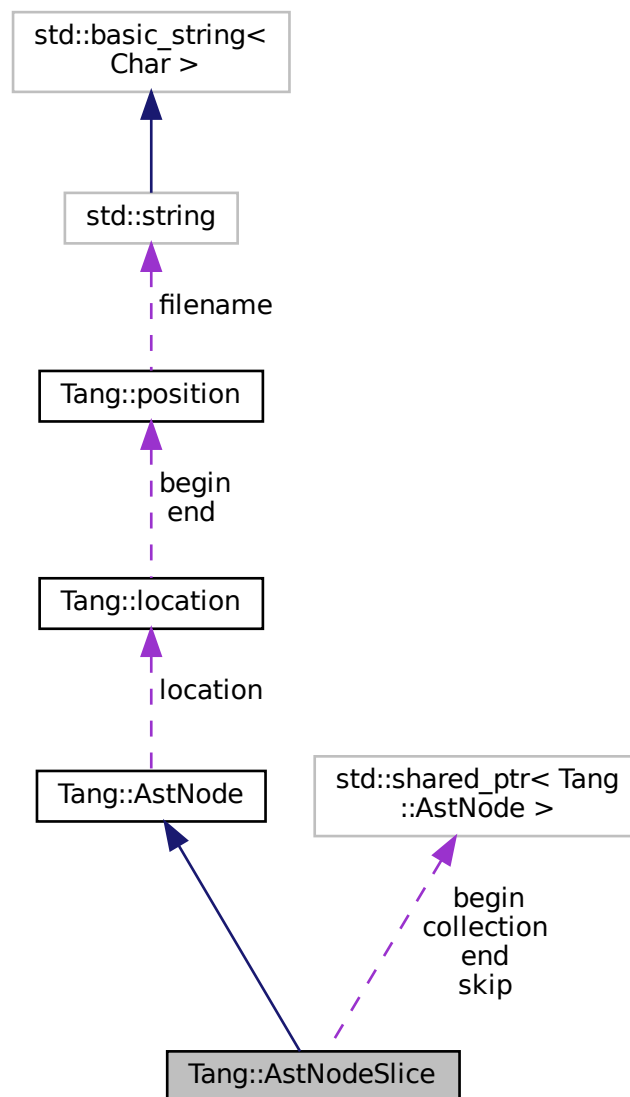
An [AstNode](#) that represents a ternary expression.

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

Inheritance diagram for Tang::AstNodeSlice:



Collaboration diagram for Tang::AstNodeSlice:



Public Types

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

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

Public Member Functions

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

The constructor.

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

Protected Attributes

- [Tang::location](#) location
The location associated with this node.

Private Attributes

- shared_ptr< [AstNode](#) > [collection](#)
The collection which will be sliced.
- shared_ptr< [AstNode](#) > [begin](#)
The begin index position of the slice.
- shared_ptr< [AstNode](#) > [end](#)
The end index position of the slice.
- shared_ptr< [AstNode](#) > [skip](#)
The skip index position of the slice.

5.23.1 Detailed Description

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

5.23.2 Member Enumeration Documentation

5.23.2.1 PreprocessState

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

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

Enumerator

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

5.23.3 Constructor & Destructor Documentation

5.23.3.1 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.

5.23.4 Member Function Documentation

5.23.4.1 compile()

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

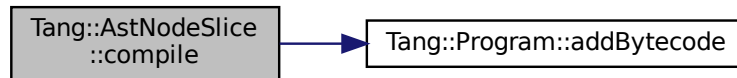
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.23.4.2 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](#).

5.23.4.3 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:

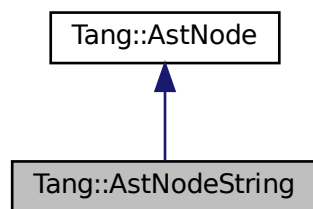
- [include/astNodeSlice.hpp](#)
- [src/astNodeSlice.cpp](#)

5.24 Tang::AstNodeString Class Reference

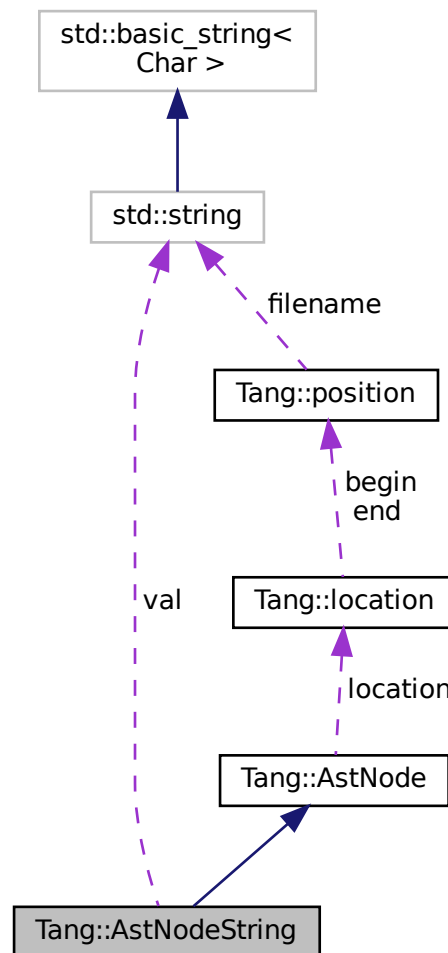
An [AstNode](#) that represents a string literal.

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

Inheritance diagram for Tang::AstNodeString:



Collaboration diagram for Tang::AstNodeString:



Public Types

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

Public Member Functions

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

Protected Attributes

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

The location associated with this node.

Private Attributes

- `std::string` `val`

The string value being stored.

5.24.1 Detailed Description

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

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 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.

5.24.4 Member Function Documentation

5.24.4.1 compile()

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

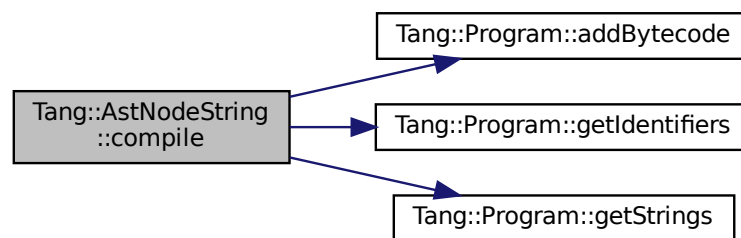
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.24.4.2 compileLiteral()

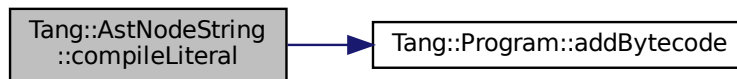
```
void AstNodeString::compileLiteral (
    Tang::Program & program ) const
```

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:



5.24.4.3 compilePreprocess()

```

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

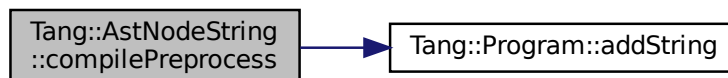
Run any preprocess analysis needed before compilation.

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:



5.24.4.4 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:

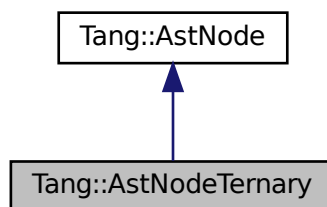
- [include/astNodeString.hpp](#)
- [src/astNodeString.cpp](#)

5.25 Tang::AstNodeTernary Class Reference

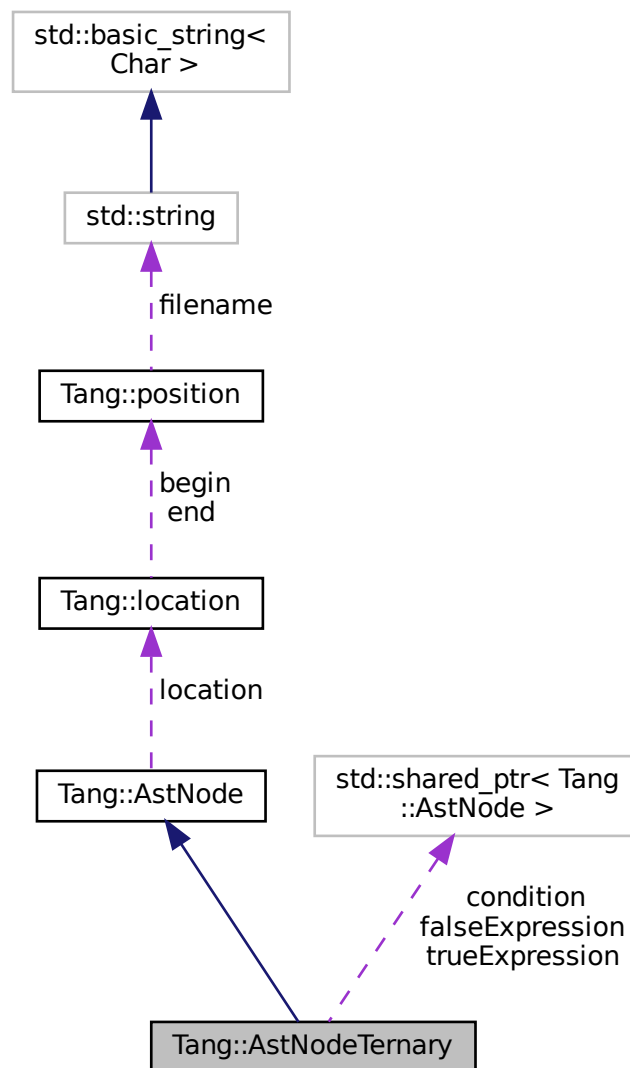
An [AstNode](#) that represents a ternary expression.

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

Inheritance diagram for Tang::AstNodeTernary:



Collaboration diagram for Tang::AstNodeTernary:



Public Types

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

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

Public Member Functions

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

The constructor.

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

Return a string that describes the contents of the node.

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

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

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

Run any preprocess analysis needed before compilation.

Protected Attributes

- [Tang::location](#) location

The location associated with this node.

Private Attributes

- shared_ptr< [AstNode](#) > condition

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

- shared_ptr< [AstNode](#) > trueExpression

The expression executed when the condition is true.

- shared_ptr< [AstNode](#) > falseExpression

The expression executed when the condition is false.

5.25.1 Detailed Description

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

5.25.2 Member Enumeration Documentation

5.25.2.1 PreprocessState

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

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

Enumerator

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

5.25.3 Constructor & Destructor Documentation

5.25.3.1 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.

5.25.4 Member Function Documentation

5.25.4.1 compile()

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

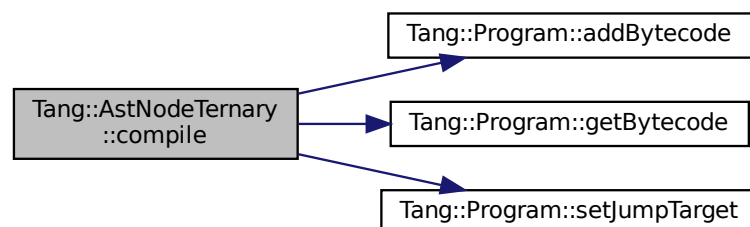
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.25.4.2 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](#).

5.25.4.3 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:

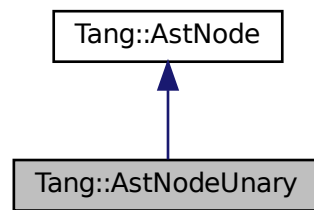
- include/[astNodeTernary.hpp](#)
- src/[astNodeTernary.cpp](#)

5.26 Tang::AstNodeUnary Class Reference

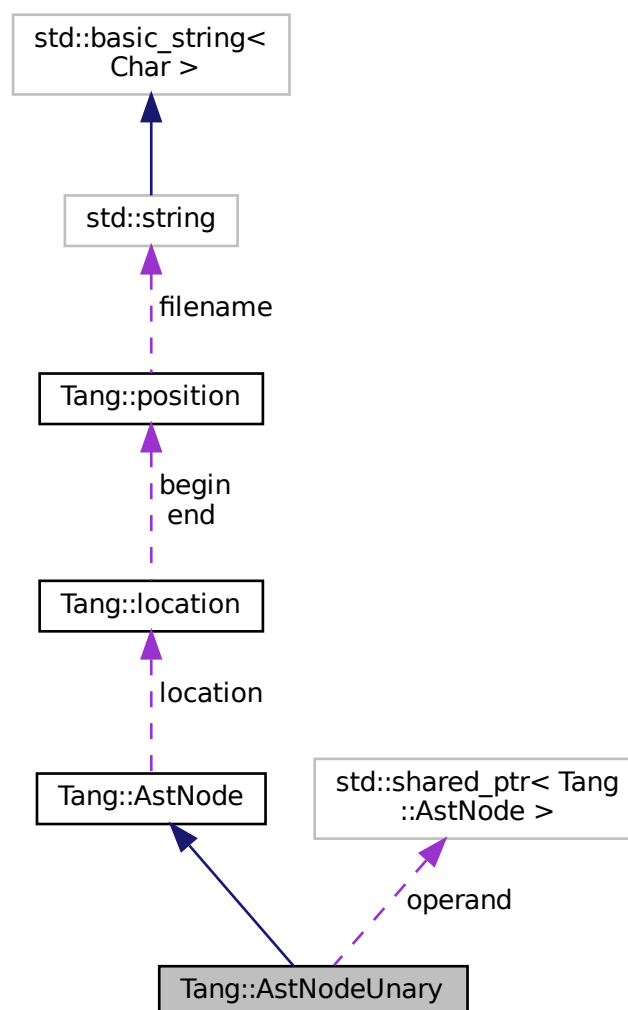
An [AstNode](#) that represents a unary negation.

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

Inheritance diagram for Tang::AstNodeUnary:



Collaboration diagram for Tang::AstNodeUnary:



Public Types

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

Public Member Functions

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

Protected Attributes

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

Private Attributes

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

5.26.1 Detailed Description

An `AstNode` that represents a unary negation.

5.26.2 Member Enumeration Documentation

5.26.2.1 Operator

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

The type of operation.

Enumerator

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

5.26.2.2 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.

5.26.3 Constructor & Destructor Documentation

5.26.3.1 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.

5.26.4 Member Function Documentation

5.26.4.1 compile()

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

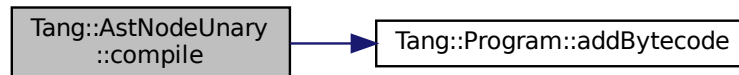
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.26.4.2 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](#).

5.26.4.3 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:

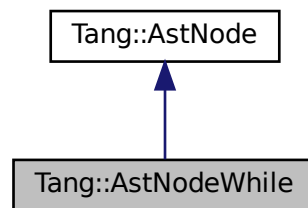
- include/[astNodeUnary.hpp](#)
- src/[astNodeUnary.cpp](#)

5.27 Tang::AstNodeWhile Class Reference

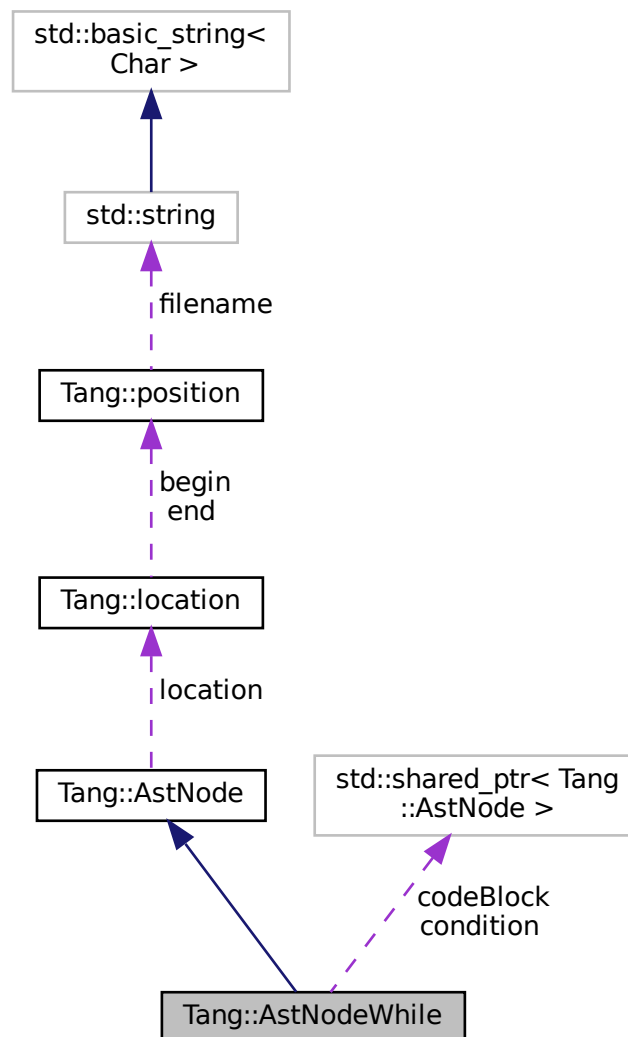
An [AstNode](#) that represents a while statement.

```
#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.

Protected Attributes

- [Tang::location](#) location
The location associated with this node.

Private Attributes

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

5.27.1 Detailed Description

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

5.27.2 Member Enumeration Documentation

5.27.2.1 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.

5.27.3 Constructor & Destructor Documentation

5.27.3.1 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.

5.27.4 Member Function Documentation

5.27.4.1 compile()

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

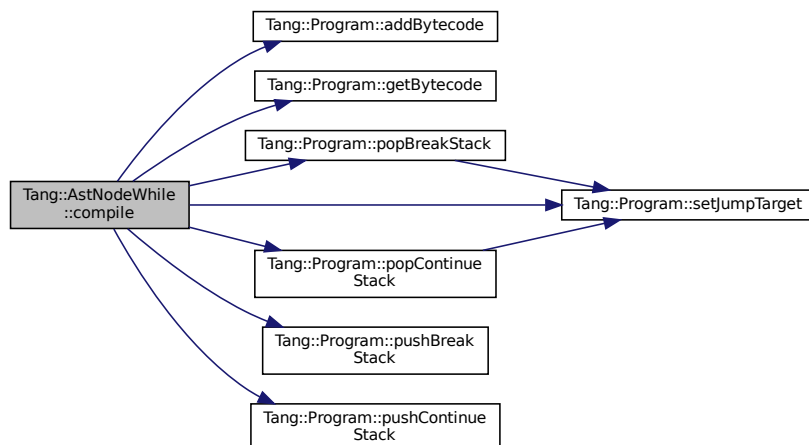
Compile the ast of the provided [Tang::Program](#).

Parameters

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

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

Here is the call graph for this function:



5.27.4.2 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](#).

5.27.4.3 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:

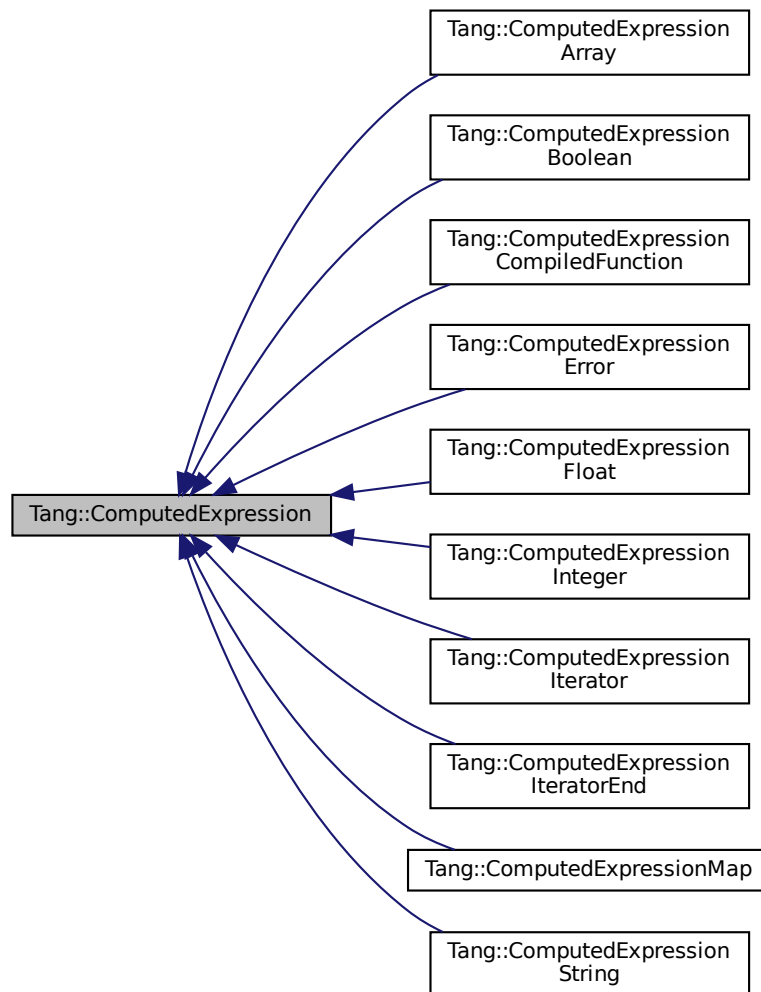
- [include/astNodeWhile.hpp](#)
- [src/astNodeWhile.cpp](#)

5.28 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

5.28.1 Detailed Description

Represents the result of a computation that has been executed.

By default, it will represent a NULL value.

5.28.2 Member Function Documentation

5.28.2.1 __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](#).

5.28.2.2 __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](#).

5.28.2.3 __assign_index()

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.28.2.4 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.28.2.5 __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](#).

5.28.2.6 __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](#).

5.28.2.7 `__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](#).

5.28.2.8 `__getIterator()`

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

Get an iterator for the expression.

Parameters

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

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

5.28.2.9 `__index()`

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.28.2.10 __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](#).

5.28.2.11 __iteratorNext()

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

Get the next iterative value.

Parameters

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

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

5.28.2.12 __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](#).

5.28.2.13 `__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](#).

5.28.2.14 `__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](#).

5.28.2.15 `__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](#).

5.28.2.16 `__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](#).

5.28.2.17 `__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](#).

5.28.2.18 __string()

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

Perform a type cast to string.

Returns

The result of the the operation.

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

5.28.2.19 __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](#).

5.28.2.20 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::ComputedExpressionMap](#), [Tang::ComputedExpressionIteratorEnd](#), [Tang::ComputedExpressionIterator](#), [Tang::ComputedExpressionInteger](#), [Tang::ComputedExpressionFloat](#), [Tang::ComputedExpressionError](#), [Tang::ComputedExpressionCompiledFunction](#), [Tang::ComputedExpressionBoolean](#), and [Tang::ComputedExpressionArray](#).

5.28.2.21 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](#).

5.28.2.22 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](#).

5.28.2.23 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.

5.28.2.24 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](#).

5.28.2.25 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](#).

5.28.2.26 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](#).

5.28.2.27 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::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

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

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

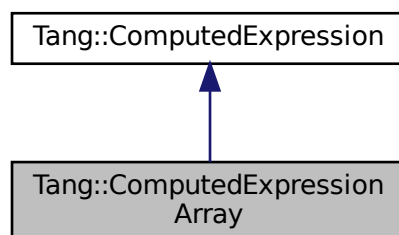
- [include/computedExpression.hpp](#)
- [src/computedExpression.cpp](#)

5.29 Tang::ComputedExpressionArray Class Reference

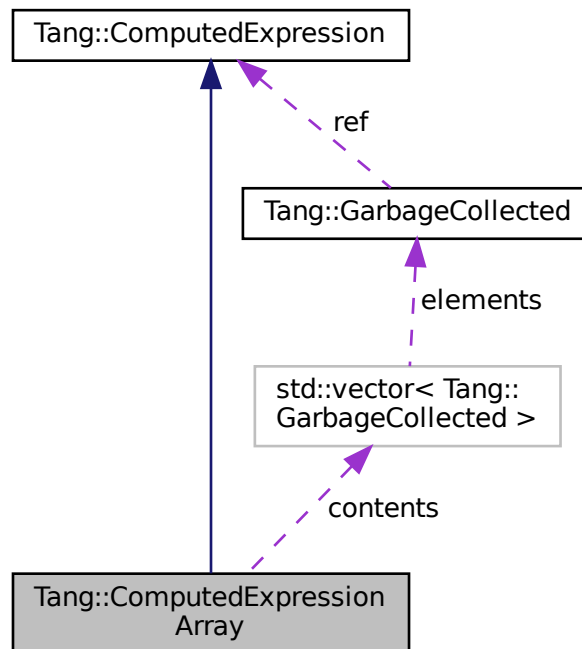
Represents an Array that is the result of a computation.

```
#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.

5.29.1 Detailed Description

Represents an Array that is the result of a computation.

5.29.2 Constructor & Destructor Documentation

5.29.2.1 ComputedExpressionArray()

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

Construct an Array result.

Parameters

<i>val</i>	The integer 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

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

Returns

The result of the operation.

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

5.29.3.2 __asCode()

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

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

Returns

A code-string representation of the computed expression.

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

5.29.3.3 __assign_index()

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

Perform an index assignment to the supplied value.

Parameters

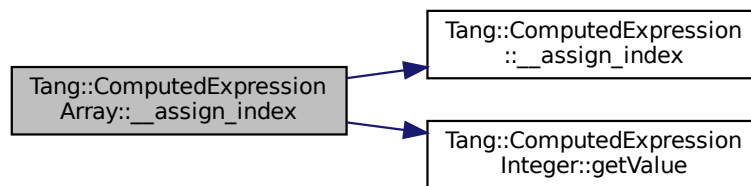
<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

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

Here is the call graph for this function:



5.29.3.4 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.29.3.5 __divide()

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

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

Parameters

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

Returns

The result of the operation.

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

5.29.3.6 `__equal()`

```
GarbageCollected 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](#).

5.29.3.7 `__float()`

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.29.3.8 `__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](#).

5.29.3.9 __index()

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

Perform an index operation.

Parameters

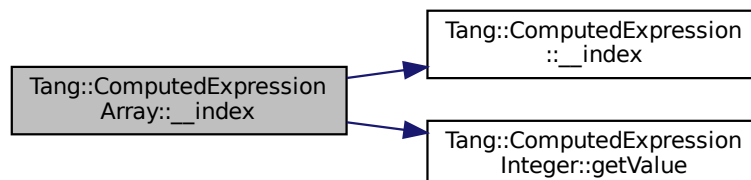
<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:



5.29.3.10 __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](#).

5.29.3.11 `__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](#).

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.29.3.17 `__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

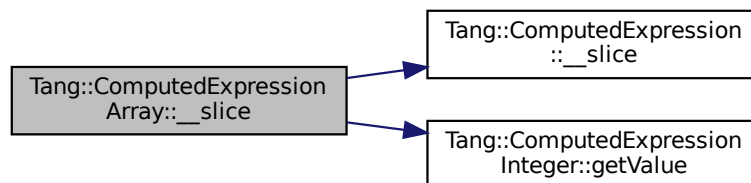
<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

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

Here is the call graph for this function:



5.29.3.18 `__string()`

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

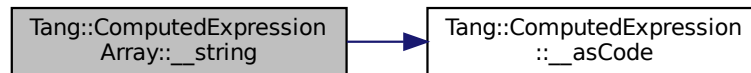
Perform a type cast to string.

Returns

The result of the the operation.

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

Here is the call graph for this function:

**5.29.3.19 __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 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](#).

5.29.3.21 `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](#).

5.29.3.22 `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 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](#).

5.29.3.28 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:

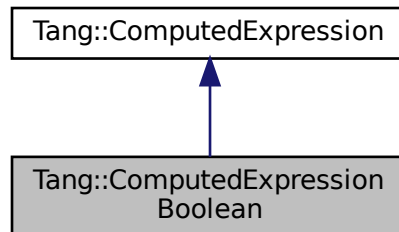
- include/[computedExpressionArray.hpp](#)
- src/[computedExpressionArray.cpp](#)

5.30 Tang::ComputedExpressionBoolean Class Reference

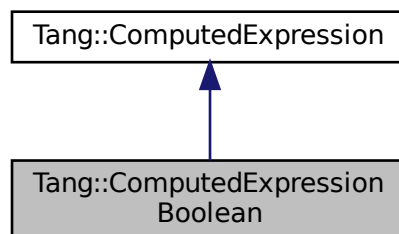
Represents an Boolean that is the result of a computation.

```
#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.

5.30.1 Detailed Description

Represents an Boolean that is the result of a computation.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 ComputedExpressionBoolean()

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

Construct an Boolean result.

Parameters

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

5.30.3 Member Function Documentation

5.30.3.1 __add()

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

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

Parameters

<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`.

5.30.3.2 `__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::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.30.3.4 `__boolean()`

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

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

Perform an equality test.

Parameters

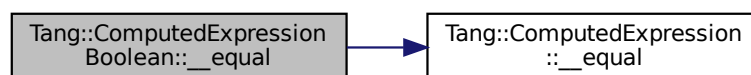
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:

5.30.3.7 `__float()`

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.30.3.8 `__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](#), [Tang::ComputedExpressionMap](#), 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](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.30.3.10 `__integer()`

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.30.3.11 `__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::ComputedExpressionMap](#), [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 ComputedExpressionBoolean::__not ( ) const [override], [virtual]
```

Compute the logical not of this value.

Returns

The result of the operation.

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

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::ComputedExpressionMap](#), [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 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](#).

5.30.3.21 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](#).

5.30.3.22 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::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.30.3.28 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:

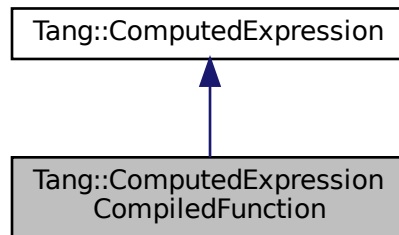
- include/computedExpressionBoolean.hpp
- src/computedExpressionBoolean.cpp

5.31 Tang::ComputedExpressionCompiledFunction Class Reference

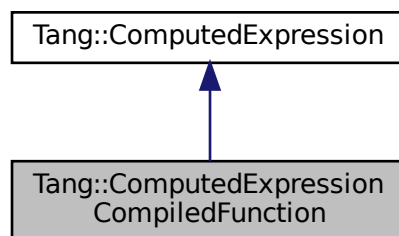
Represents a Compiled Function declared in the script.

```
#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.

5.31.1 Detailed Description

Represents a Compiled Function declared in the script.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 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.

5.31.3 Member Function Documentation

5.31.3.1 __add()

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

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

Parameters

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

Returns

The result of the operation.

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

5.31.3.2 __asCode()

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

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

Returns

A code-string representation of the computed expression.

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

5.31.3.3 __assign_index()

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.31.3.4 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.31.3.5 __divide()

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

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

Parameters

<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](#).

5.31.3.6 __equal()

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

Perform an equality test.

Parameters

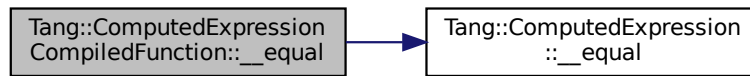
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.31.3.7 __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](#).

5.31.3.8 __getIterator()

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

Get an iterator for the expression.

Parameters

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

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

5.31.3.9 __index()

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.31.3.10 __integer()

```
GarbageCollected 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](#).

5.31.3.11 __iteratorNext()

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

Get the next iterative value.

Parameters

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

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

5.31.3.12 __lessThan()

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

Compute the "less than" comparison.

Parameters

<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](#).

5.31.3.13 `__modulo()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.31.3.14 `__multiply()`

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

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

Parameters

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

Returns

The result of the operation.

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

5.31.3.15 __negative()

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.31.3.16 __not()

```
GarbageCollected 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](#).

5.31.3.17 __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 ComputedExpression::__string ( ) const [virtual], [inherited]
```

Perform a type cast to string.

Returns

The result of the the operation.

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

5.31.3.19 __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](#).

5.31.3.20 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](#).

5.31.3.21 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]

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

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

Parameters

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

Returns

True if equal, false if not.

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

5.31.3.23 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::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.31.3.28 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:

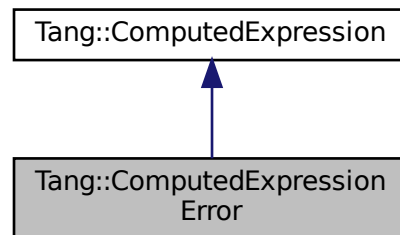
- [include/computedExpressionCompiledFunction.hpp](#)
- [src/computedExpressionCompiledFunction.cpp](#)

5.32 Tang::ComputedExpressionError Class Reference

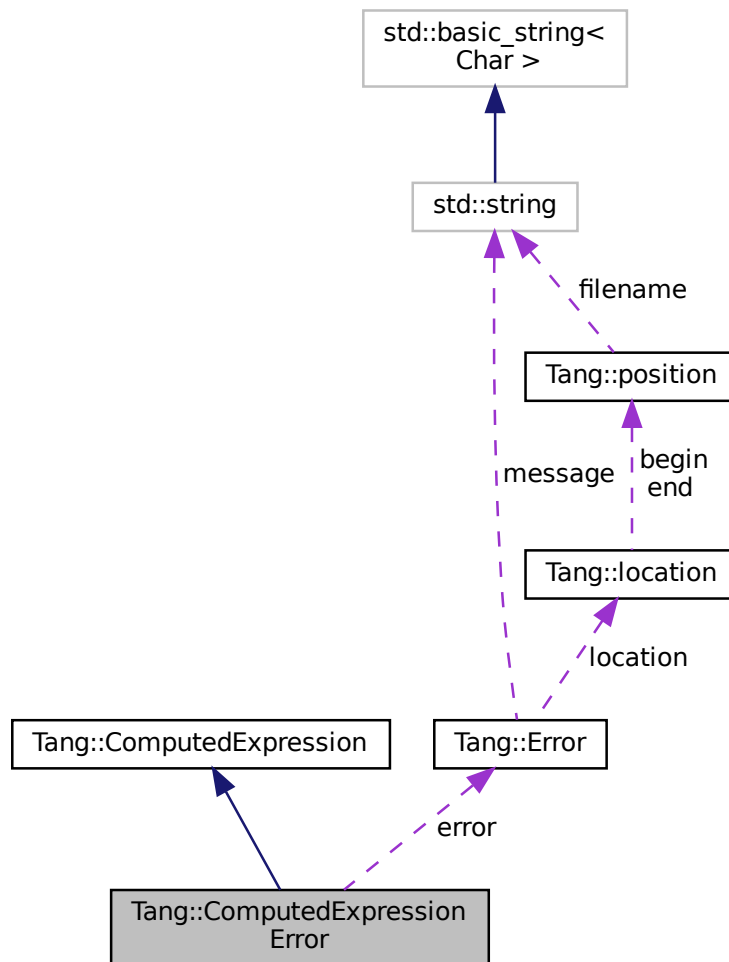
Represents a Runtime [Error](#).

```
#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.

5.32.1 Detailed Description

Represents a Runtime [Error](#).

5.32.2 Constructor & Destructor Documentation

5.32.2.1 ComputedExpressionError()

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

Construct a Runtime [Error](#).

Parameters

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

5.32.3 Member Function Documentation

5.32.3.1 __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](#).

5.32.3.2 __asCode()

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

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

Returns

A code-string representation of the computed expression.

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

5.32.3.3 __assign_index()

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.32.3.4 __boolean()

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

5.32.3.6 `__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](#).

5.32.3.7 `__float()`

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.32.3.8 `__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](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.32.3.9 __index()

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.32.3.10 __integer()

```
GarbageCollected ComputedExpressionError::__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::ComputedExpressionMap](#), [Tang::ComputedExpressionIterator](#), and [Tang::ComputedExpressionArray](#).

5.32.3.12 __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](#).

5.32.3.13 __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](#).

5.32.3.14 __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](#).

5.32.3.15 __negative()

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.32.3.16 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

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

Perform a type cast to string.

Returns

The result of the the operation.

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

5.32.3.19 `__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](#).

5.32.3.20 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](#).

5.32.3.21 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](#).

5.32.3.22 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](#).

5.32.3.23 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.

5.32.3.24 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](#).

5.32.3.25 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](#).

5.32.3.26 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](#).

5.32.3.27 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::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.32.3.28 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:

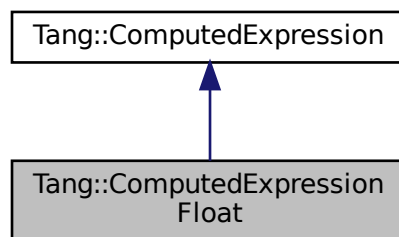
- [include/computedExpressionError.hpp](#)
- [src/computedExpressionError.cpp](#)

5.33 Tang::ComputedExpressionFloat Class Reference

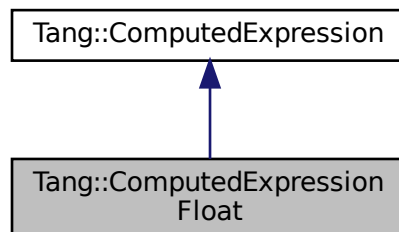
Represents a Float that is the result of a computation.

```
#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)

- Perform an index assignment to the supplied value.*
- virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const
Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __index` (const `GarbageCollected` &index) const
Perform an index operation.
- virtual `GarbageCollected __slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const
Perform a slice operation.
- 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::float_t val`
The float value.

5.33.1 Detailed Description

Represents a Float that is the result of a computation.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 ComputedExpressionFloat()

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

Construct a Float result.

Parameters

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

5.33.3 Member Function Documentation

5.33.3.1 __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:



5.33.3.2 __asCode()

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

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

Returns

A code-string representation of the computed expression.

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

5.33.3.3 __assign_index()

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.33.3.4 __boolean()

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

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

Parameters

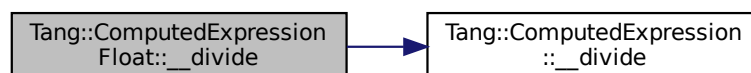
<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.33.3.6 `__equal()`

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

Perform an equality test.

Parameters

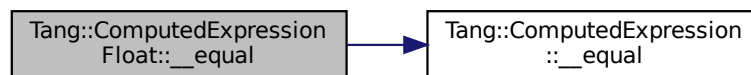
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:

5.33.3.7 `__float()`

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.33.3.8 `__getIterator()`

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

Get an iterator for the expression.

Parameters

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

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

5.33.3.9 __index()

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.33.3.10 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.33.3.11 __iteratorNext()

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

Get the next iterative value.

Parameters

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

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

5.33.3.12 __lessThan()

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

Compute the "less than" comparison.

Parameters

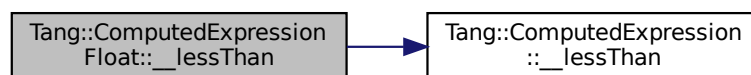
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.33.3.13 __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](#).

5.33.3.14 __multiply()

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

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

Parameters

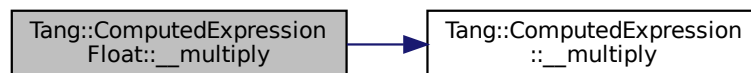
<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:

**5.33.3.15 __negative()**

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.33.3.16 `__not()`

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

Compute the logical not of this value.

Returns

The result of the operation.

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

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

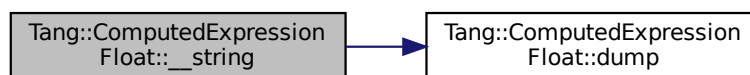
Perform a type cast to string.

Returns

The result of the the operation.

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

Here is the call graph for this function:

**5.33.3.19 __subtract()**

```

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

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

Parameters

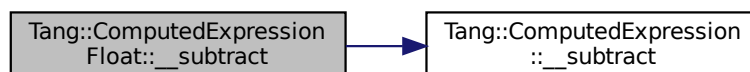
<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.33.3.20 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](#).

5.33.3.21 getValue()

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

Helper function to get the value associated with this expression.

Returns

The value associated with this expression.

5.33.3.22 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](#).

5.33.3.23 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 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](#).

5.33.3.27 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](#).

5.33.3.28 isCopyNeeded()

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

Determine whether or not a copy is needed.

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

Returns

Whether or not a copy is needed.

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

5.33.3.29 makeCopy()

`GarbageCollected` `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:

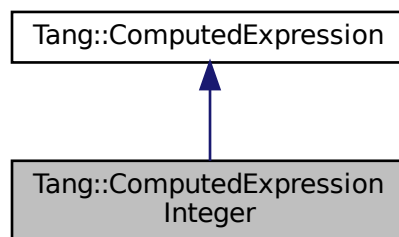
- `include/computedExpressionFloat.hpp`
- `src/computedExpressionFloat.cpp`

5.34 Tang::ComputedExpressionInteger Class Reference

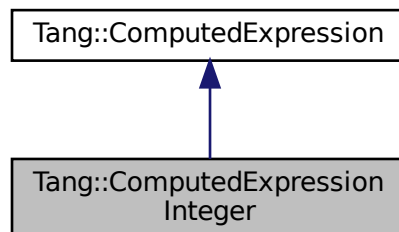
Represents an Integer that is the result of a computation.

```
#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.

5.34.1 Detailed Description

Represents an Integer that is the result of a computation.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 ComputedExpressionInteger()

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

Construct an Integer result.

Parameters

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

5.34.3 Member Function Documentation

5.34.3.1 __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:

**5.34.3.2 __asCode()**

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

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

Returns

A code-string representation of the computed expression.

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

5.34.3.3 __assign_index()

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.34.3.4 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.34.3.5 __divide()

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

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

Parameters

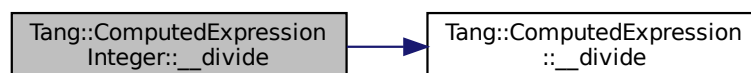
<i>rhs</i>	The GarbageCollected value to divide this by.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.34.3.6 `__equal()`

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

Perform an equality test.

Parameters

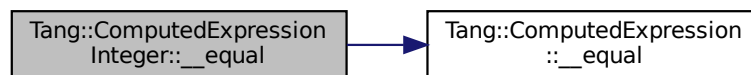
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.34.3.7 `__float()`

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.34.3.8 `__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](#), [Tang::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.34.3.9 __index()

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.34.3.10 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.34.3.11 __iteratorNext()

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

Get the next iterative value.

Parameters

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

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

5.34.3.12 __lessThan()

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

Compute the "less than" comparison.

Parameters

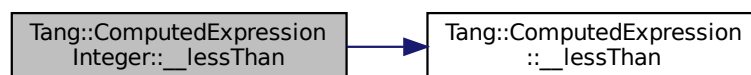
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:

**5.34.3.13 __modulo()**

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

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

Parameters

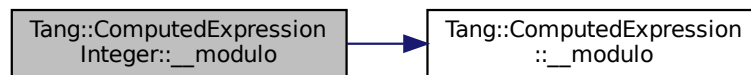
<i>rhs</i>	The GarbageCollected value to modulo this by.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:

**5.34.3.14 __multiply()**

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

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

Parameters

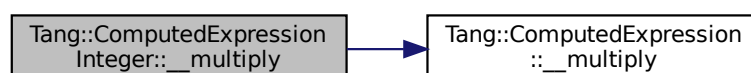
<i>rhs</i>	The GarbageCollected value to multiply to this.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.34.3.15 __negative()

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.34.3.16 __not()

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

Compute the logical not of this value.

Returns

The result of the operation.

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

5.34.3.17 __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 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:



5.34.3.19 __subtract()

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

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

Parameters

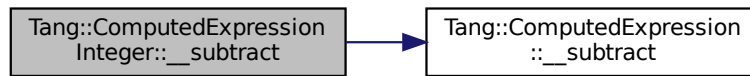
<i>rhs</i>	The GarbageCollected value to subtract from this.
------------	---

Returns

The result of the operation.

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

Here is the call graph for this function:



5.34.3.20 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](#).

5.34.3.21 getValue()

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

Helper function to get the value associated with this expression.

Returns

The value associated with this expression.

5.34.3.22 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](#).

5.34.3.23 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](#).

5.34.3.24 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.

5.34.3.25 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](#).

5.34.3.26 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](#).

5.34.3.27 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](#).

5.34.3.28 isCopyNeeded()

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

Determine whether or not a copy is needed.

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

Returns

Whether or not a copy is needed.

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

5.34.3.29 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:

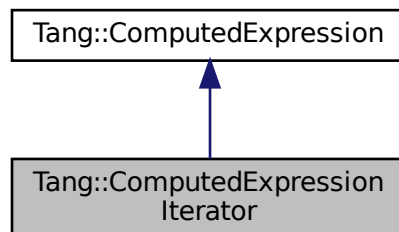
- [include/computedExpressionInteger.hpp](#)
- [src/computedExpressionInteger.cpp](#)

5.35 Tang::ComputedExpressionIterator Class Reference

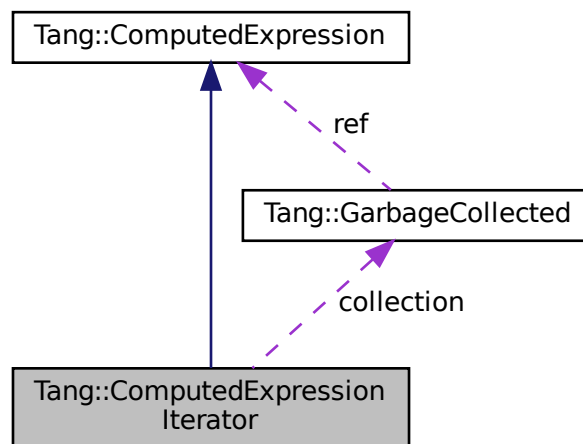
Represents an Iterator that is the result of a computation.

```
#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.

5.35.1 Detailed Description

Represents an Iterator that is the result of a computation.

5.35.2 Constructor & Destructor Documentation

5.35.2.1 ComputedExpressionIterator()

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

Construct an iterator result.

Parameters

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

5.35.3 Member Function Documentation

5.35.3.1 __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](#).

5.35.3.2 __asCode()

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

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

Returns

A code-string representation of the computed expression.

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

5.35.3.3 __assign_index()

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.35.3.4 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.35.3.5 __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](#).

5.35.3.6 `__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](#).

5.35.3.7 `__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](#).

5.35.3.8 `__getIterator()`

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

Get an iterator for the expression.

Parameters

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

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

5.35.3.9 __index()

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.35.3.10 __integer()

```
GarbageCollected 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](#).

5.35.3.11 __iteratorNext()

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

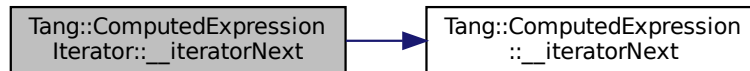
Get the next iterative value.

Parameters

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

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

Here is the call graph for this function:



5.35.3.12 __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](#).

5.35.3.13 __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](#).

5.35.3.14 __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](#).

5.35.3.15 __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](#).

5.35.3.16 __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](#).

5.35.3.17 __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](#).

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

5.35.3.19 __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](#).

5.35.3.20 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](#).

5.35.3.21 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](#).

5.35.3.22 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](#).

5.35.3.23 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.

5.35.3.24 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](#).

5.35.3.25 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](#).

5.35.3.26 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](#).

5.35.3.27 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::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.35.3.28 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::ComputedExpressionMap`, `Tang::ComputedExpressionInteger`, `Tang::ComputedExpressionFloat`, `Tang::ComputedExpressionError`, `Tang::ComputedExpressionCompiledFunction`, `Tang::ComputedExpressionBoolean`, and `Tang::ComputedExpressionArray`.

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

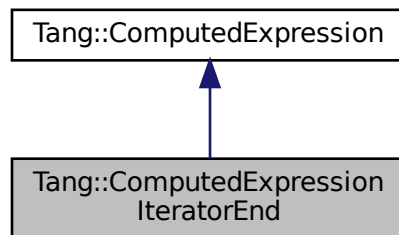
- `include/computedExpressionIterator.hpp`
- `src/computedExpressionIterator.cpp`

5.36 Tang::ComputedExpressionIteratorEnd Class Reference

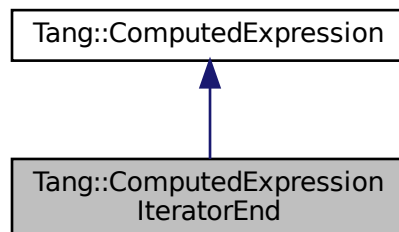
Represents that a collection has no more values through which to iterate.

```
#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.

5.36.1 Detailed Description

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

5.36.2 Member Function Documentation

5.36.2.1 [__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](#).

5.36.2.2 [__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](#).

5.36.2.3 `__assign_index()`

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.36.2.4 `__boolean()`

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.36.2.5 `__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](#).

5.36.2.6 __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](#).

5.36.2.7 __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](#).

5.36.2.8 __getIterator()

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

Get an iterator for the expression.

Parameters

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

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

5.36.2.9 `__index()`

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

Perform an index operation.

Parameters

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

Returns

The result of the operation.

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

5.36.2.10 `__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](#).

5.36.2.11 `__iteratorNext()`

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

Get the next iterative value.

Parameters

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

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

5.36.2.12 __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](#).

5.36.2.13 __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](#).

5.36.2.14 __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](#).

5.36.2.15 __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](#).

5.36.2.16 __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](#).

5.36.2.17 __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](#).

5.36.2.18 `__string()`

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

Perform a type cast to string.

Returns

The result of the the operation.

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

5.36.2.19 `__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](#).

5.36.2.20 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](#).

5.36.2.21 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](#).

5.36.2.22 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](#).

5.36.2.23 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.

5.36.2.24 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](#).

5.36.2.25 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](#).

5.36.2.26 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](#).

5.36.2.27 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::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.36.2.28 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::ComputedExpressionMap`, `Tang::ComputedExpressionInteger`, `Tang::ComputedExpressionFloat`, `Tang::ComputedExpressionError`, `Tang::ComputedExpressionCompiledFunction`, `Tang::ComputedExpressionBoolean`, and `Tang::ComputedExpressionArray`.

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

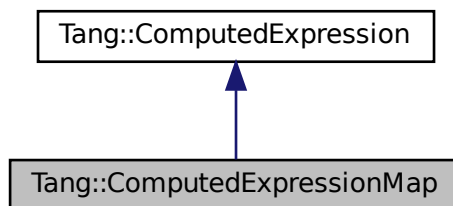
- `include/computedExpressionIteratorEnd.hpp`
- `src/computedExpressionIteratorEnd.cpp`

5.37 Tang::ComputedExpressionMap Class Reference

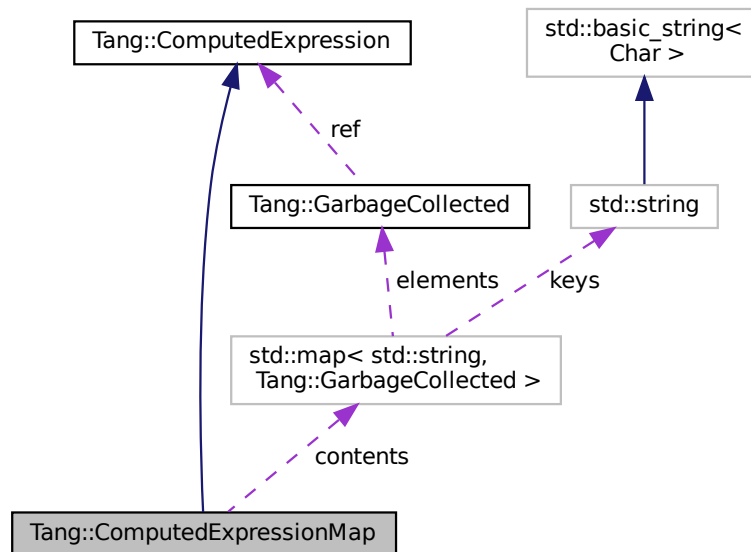
Represents an Map that is the result of a computation.

```
#include <computedExpressionMap.hpp>
```

Inheritance diagram for Tang::ComputedExpressionMap:



Collaboration diagram for Tang::ComputedExpressionMap:



Public Member Functions

- [ComputedExpressionMap](#) (std::map< std::string, [Tang::GarbageCollected](#) > [contents](#))
Construct an Map result.
- virtual std::string [dump](#) () const override
Output the contents of the [ComputedExpression](#) as a string.
- virtual bool [isCopyNeeded](#) () const override
Determine whether or not a copy is needed.
- [GarbageCollected](#) [makeCopy](#) () const override
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).
- virtual [GarbageCollected](#) [__index](#) (const [GarbageCollected](#) &index) const override
Perform an index operation.
- virtual [GarbageCollected](#) [__getIterator](#) (const [GarbageCollected](#) &collection) const override
Get an iterator for the expression.
- virtual [GarbageCollected](#) [__iteratorNext](#) (size_t index) const override
Get the next iterative value.
- virtual [GarbageCollected](#) [__assign_index](#) (const [GarbageCollected](#) &index, const [GarbageCollected](#) &value) override
Perform an index assignment to the supplied value.
- virtual [GarbageCollected](#) [__string](#) () const override
Perform a type cast to string.
- virtual [GarbageCollected](#) [__boolean](#) () const override
Perform a type cast to boolean.
- virtual std::string [__asCode](#) () const
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.
- virtual bool [is_equal](#) (const [Tang::integer_t](#) &val) const

- virtual bool `is_equal` (const `Tang::float_t` &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const bool &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const string &val) const
Check whether or not the computed expression is equal to another value.
- virtual bool `is_equal` (const `Error` &val) const
Check whether or not the computed expression is equal to another value.
- virtual 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` `__slice` (const `GarbageCollected` &begin, const `GarbageCollected` &end, const `GarbageCollected` &skip) const
Perform a slice operation.
- virtual `GarbageCollected` `__integer` () const
Perform a type cast to integer.
- virtual `GarbageCollected` `__float` () const
Perform a type cast to float.

Private Attributes

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

5.37.1 Detailed Description

Represents an Map that is the result of a computation.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 ComputedExpressionMap()

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

Construct an Map result.

Parameters

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

5.37.3 Member Function Documentation

5.37.3.1 __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](#).

5.37.3.2 __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](#).

5.37.3.3 __assign_index()

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

Perform an index assignment to the supplied value.

Parameters

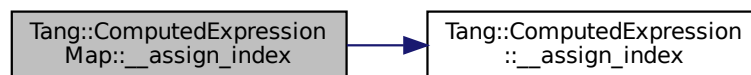
<i>index</i>	The index to which the value should be applied.
<i>value</i>	The value to store.

Returns

The result of the operation.

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

Here is the call graph for this function:



5.37.3.4 __boolean()

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

Perform a type cast to boolean.

Returns

The result of the the operation.

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

5.37.3.5 __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](#).

5.37.3.6 `__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](#).

5.37.3.7 `__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](#).

5.37.3.8 `__getIterator()`

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

Get an iterator for the expression.

Parameters

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

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

5.37.3.9 `__index()`

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

Perform an index operation.

Parameters

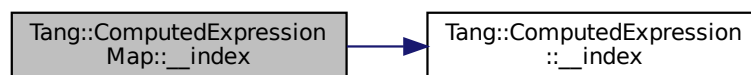
<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:

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

5.37.3.11 `__iteratorNext()`

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

Get the next iterative value.

Parameters

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

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

5.37.3.12 `__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](#).

5.37.3.13 `__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](#).

5.37.3.14 __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](#).

5.37.3.15 __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](#).

5.37.3.16 __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](#).

5.37.3.17 `__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](#).

5.37.3.18 `__string()`

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

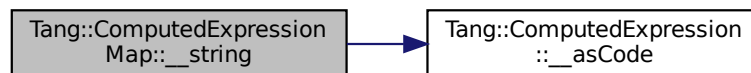
Perform a type cast to string.

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.37.3.19 `__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](#).

5.37.3.20 dump()

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

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

Returns

A string representation of the computed expression.

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

5.37.3.21 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](#).

5.37.3.22 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](#).

5.37.3.23 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.

5.37.3.24 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](#).

5.37.3.25 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](#).

5.37.3.26 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](#).

5.37.3.27 isCopyNeeded()

```
bool ComputedExpressionMap::isCopyNeeded ( ) const [override], [virtual]
```

Determine whether or not a copy is needed.

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

Returns

Whether or not a copy is needed.

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

5.37.3.28 makeCopy()

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

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

Returns

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

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

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

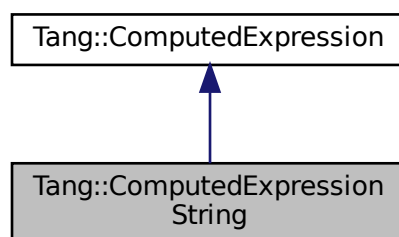
- include/computedExpressionMap.hpp
- src/computedExpressionMap.cpp

5.38 Tang::ComputedExpressionString Class Reference

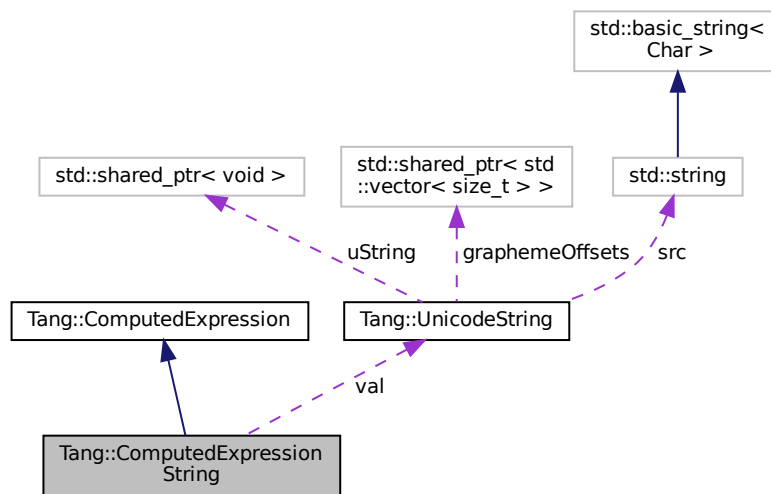
Represents a String that is the result of a computation.

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

Inheritance diagram for Tang::ComputedExpressionString:



Collaboration diagram for Tang::ComputedExpressionString:



Public Member Functions

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

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

5.38.1 Detailed Description

Represents a String that is the result of a computation.

5.38.2 Constructor & Destructor Documentation

5.38.2.1 ComputedExpressionString()

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

Construct a String result.

Parameters

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

5.38.3 Member Function Documentation

5.38.3.1 __add()

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

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

Parameters

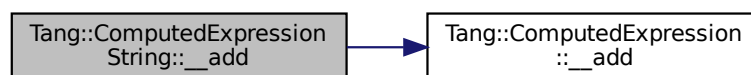
<i>rhs</i>	The GarbageCollected value to add to this.
------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:



5.38.3.2 __asCode()

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

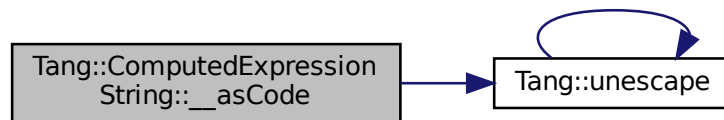
Output the contents of the [ComputedExpression](#) as a string similar to how it would be represented as code.

Returns

A code-string representation of the computed expression.

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

Here is the call graph for this function:

**5.38.3.3 __assign_index()**

```

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

Perform an index assignment to the supplied value.

Parameters

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

Returns

The result of the operation.

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

5.38.3.4 __boolean()

```

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

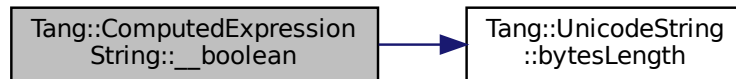
Perform a type cast to boolean.

Returns

The result of the the operation.

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

Here is the call graph for this function:

**5.38.3.5 __divide()**

```

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

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

Parameters

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

Returns

The result of the operation.

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

5.38.3.6 __equal()

```

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

Perform an equality test.

Parameters

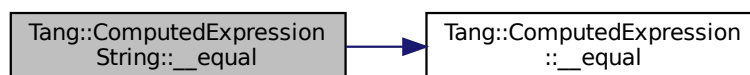
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:

**5.38.3.7 __float()**

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

Perform a type cast to float.

Returns

The result of the the operation.

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

5.38.3.8 __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](#).

5.38.3.9 __index()

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

Perform an index operation.

Parameters

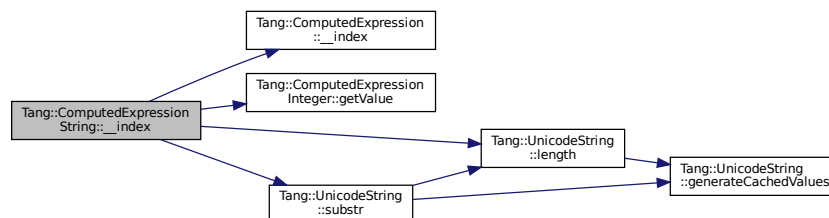
<i>index</i>	The index expression provided by the script.
--------------	--

Returns

The result of the operation.

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

Here is the call graph for this function:



5.38.3.10 __integer()

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

Perform a type cast to integer.

Returns

The result of the the operation.

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

5.38.3.11 __iteratorNext()

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

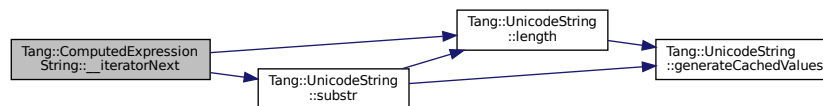
Get the next iterative value.

Parameters

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

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

Here is the call graph for this function:



5.38.3.12 __lessThan()

```

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

Compute the "less than" comparison.

Parameters

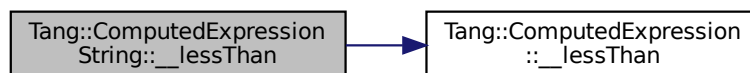
<i>rhs</i>	The GarbageCollected value to compare against.
------------	--

Returns

The result of the the operation.

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

Here is the call graph for this function:



5.38.3.13 __modulo()

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

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

Parameters

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

Returns

The result of the operation.

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

5.38.3.14 __multiply()

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

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

Parameters

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

Returns

The result of the operation.

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

5.38.3.15 __negative()

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

Compute the result of negating this value.

Returns

The result of the operation.

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

5.38.3.16 `__not()`

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

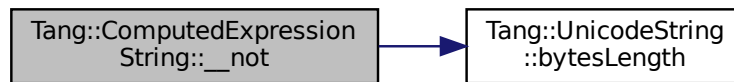
Compute the logical not of this value.

Returns

The result of the operation.

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

Here is the call graph for this function:

**5.38.3.17** `__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

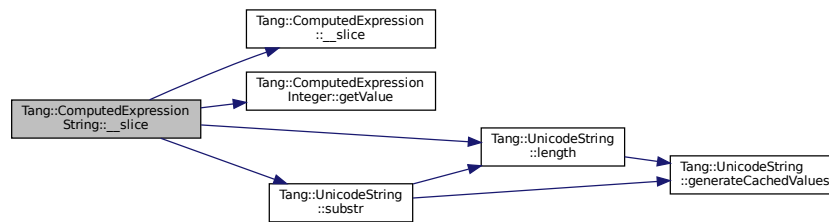
<i>begin</i>	The begin index expression provided by the script.
<i>end</i>	The end index expression provided by the script.
<i>skip</i>	The skip index expression provided by the script.

Returns

The result of the operation.

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

Here is the call graph for this function:



5.38.3.18 `__string()`

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

Perform a type cast to string.

Returns

The result of the the operation.

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

5.38.3.19 `__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](#).

5.38.3.20 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](#).

5.38.3.21 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

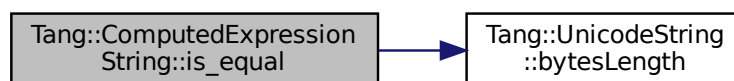
<i>val</i>	The value to compare against.
------------	-------------------------------

Returns

True if equal, false if not.

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

Here is the call graph for this function:

**5.38.3.22 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](#).

5.38.3.23 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.

5.38.3.24 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](#).

5.38.3.25 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](#).

5.38.3.26 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](#).

5.38.3.27 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::ComputedExpressionMap](#), and [Tang::ComputedExpressionArray](#).

5.38.3.28 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:

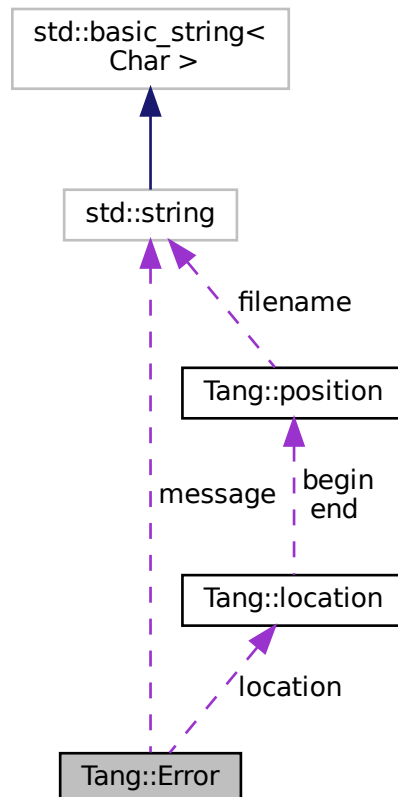
- `include/computedExpressionString.hpp`
- `src/computedExpressionString.cpp`

5.39 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.

5.39.1 Detailed Description

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

5.39.2 Constructor & Destructor Documentation

5.39.2.1 [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.
----------------	--------------------------------

5.39.2.2 [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.

5.39.3 Friends And Related Function Documentation

5.39.3.1 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:

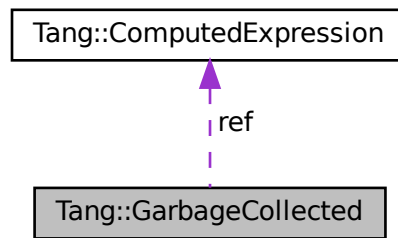
- [include/error.hpp](#)
- [src/error.cpp](#)

5.40 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.

5.40.1 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.

5.40.2 Constructor & Destructor Documentation

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

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

Copy Constructor.

Parameters

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

5.40.2.2 [GarbageCollected\(\)](#) [2/3]

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

Move Constructor.

Parameters

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

5.40.2.3 [~GarbageCollected\(\)](#)

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

Destructor.

Clean up the tracked object, if appropriate.

5.40.2.4 `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.
-----------------	---

5.40.3 Member Function Documentation

5.40.3.1 `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.

5.40.3.2 `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:

**5.40.3.3 makeCopy()**

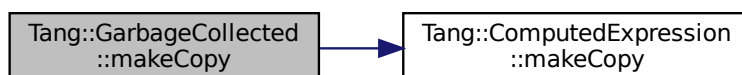
```
GarbageCollected GarbageCollected::makeCopy ( ) const
```

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:

**5.40.3.4 operator"!")()**

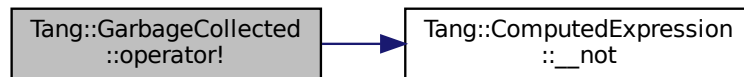
```
GarbageCollected GarbageCollected::operator! ( ) const
```

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

Returns

The result of the operation.

Here is the call graph for this function:

**5.40.3.5 operator"!="()**

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

Perform a != between two `GarbageCollected` values.

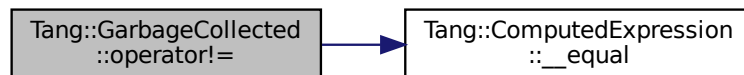
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:

**5.40.3.6 operator%()**

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

Perform a modulo between two `GarbageCollected` values.

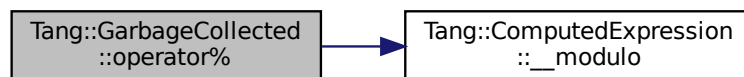
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.40.3.7 operator*() [1/2]

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

Access the tracked object.

Returns

A reference to the tracked object.

5.40.3.8 operator*() [2/2]

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

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

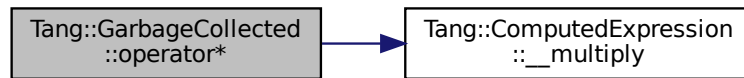
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.40.3.9 `operator+()`

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

Perform an addition between two `GarbageCollected` values.

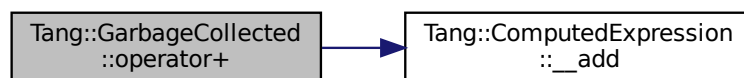
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



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

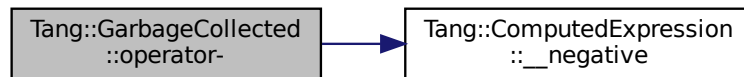
```
GarbageCollected GarbageCollected::operator- ( ) const
```

Perform a negation on the `GarbageCollected` value.

Returns

The result of the operation.

Here is the call graph for this function:

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

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

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

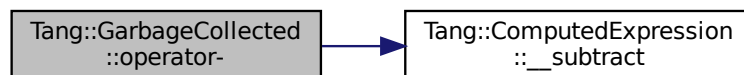
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:

**5.40.3.12 operator->()**

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

Access the tracked object as a pointer.

Returns

A pointer to the tracked object.

5.40.3.13 operator/()

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

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

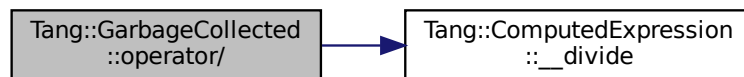
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:

**5.40.3.14 operator<()**

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

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

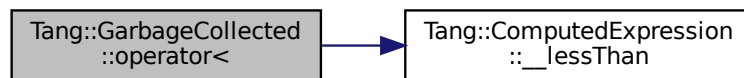
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.40.3.15 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.

5.40.3.16 operator=() [1/2]

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

Copy Assignment.

Parameters

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

5.40.3.17 operator=() [2/2]

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

Move Assignment.

Parameters

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

5.40.3.18 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.

5.40.3.19 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.

5.40.3.20 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.

5.40.3.21 `operator==()` [4/8]

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

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

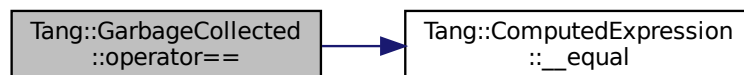
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:

5.40.3.22 `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.

5.40.3.23 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.

5.40.3.24 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.

5.40.3.25 operator==() [8/8]

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

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

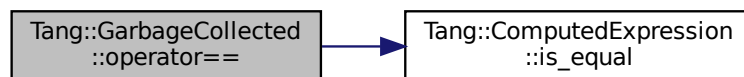
Parameters

<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:

5.40.3.26 `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.

5.40.3.27 `operator>=()`

```

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

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

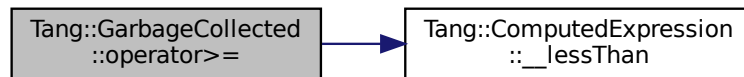
Parameters

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

Returns

The result of the operation.

Here is the call graph for this function:



5.40.4 Friends And Related Function Documentation

5.40.4.1 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:

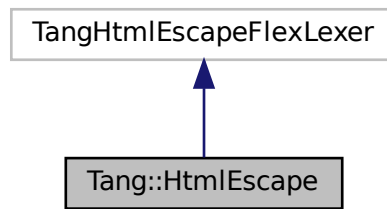
- [include/garbageCollected.hpp](#)
- [src/garbageCollected.cpp](#)

5.41 Tang::HtmlEscape Class Reference

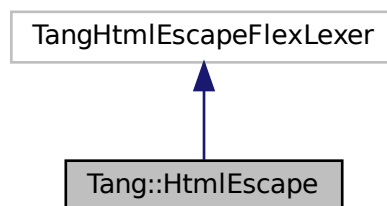
The Flex lexer class for the main Tang language.

```
#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.

5.41.1 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.

5.41.2 Constructor & Destructor Documentation

5.41.2.1 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)

5.41.3 Member Function Documentation

5.41.3.1 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:

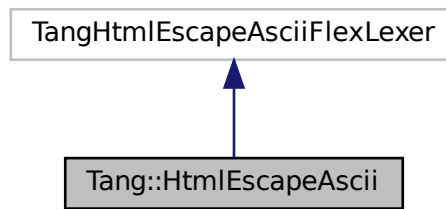
- include/[htmlEscape.hpp](#)

5.42 Tang::HtmlEscapeAscii Class Reference

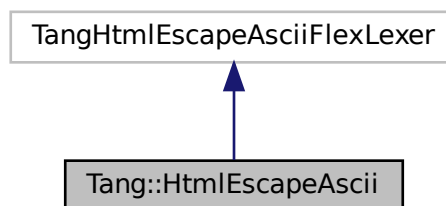
The Flex lexer class for the main Tang language.

```
#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.

5.42.1 Detailed Description

The Flex lexer class for the main Tang language.

Flex requires that our lexer class inherit from `yyFlexLexer`, an "intermediate" class whose real name is "`TangTang↔FlexLexer`". We are subclassing it so that we can override the return type of [get_next_token\(\)](#), for compatibility with Bison 3 tokens.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 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)

5.42.3 Member Function Documentation

5.42.3.1 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:

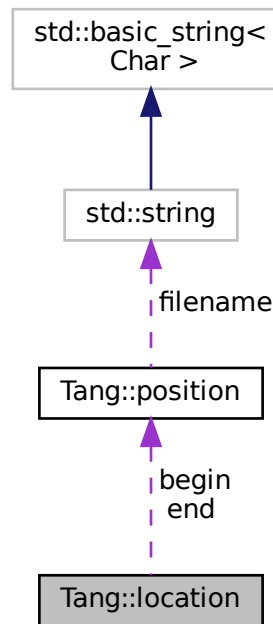
- include/[htmlEscapeAscii.hpp](#)

5.43 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.

5.43.1 Detailed Description

Two points in a source file.

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

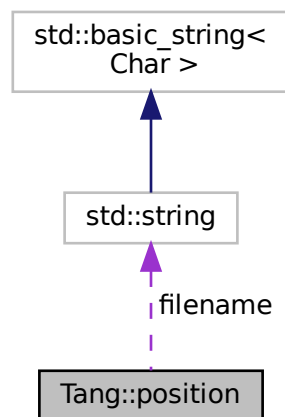
- build/generated/[location.hh](#)

5.44 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.

5.45.1 Detailed Description

Represents a compiled script or template that may be executed.

5.45.2 Member Enumeration Documentation

5.45.2.1 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.

5.45.3 Constructor & Destructor Documentation

5.45.3.1 Program()

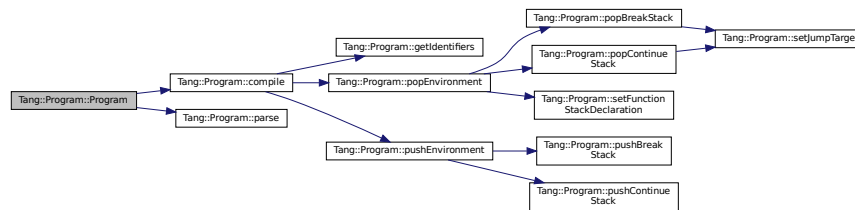
```
Program::Program (
    std::string code,
    Program::CodeType codeType )
```

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:



5.45.4 Member Function Documentation

5.45.4.1 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.
-----------------	---

5.45.4.2 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.

5.45.4.3 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.
-----------------	--

5.45.4.4 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.

5.45.4.5 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.
-------------	----------------------

5.45.4.6 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.

5.45.4.7 dumpBytecode()

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

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

Returns

A string containing the Opcode representation.

5.45.4.8 execute()

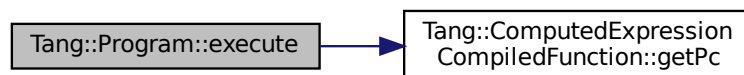
```
Program & Program::execute ( )
```

Execute the program's Bytecode, and return the current [Program](#) object.

Returns

The current [Program](#) object.

Here is the call graph for this function:

**5.45.4.9 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.

5.45.4.10 getBytecode()

```
const Bytecode & Program::getBytecode ( )
```

Get the Bytecode vector.

Returns

The Bytecode vector.

5.45.4.11 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.

5.45.4.12 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.

5.45.4.13 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.

5.45.4.14 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.

5.45.4.15 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.

5.45.4.16 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:



5.45.4.17 popContinueStack()

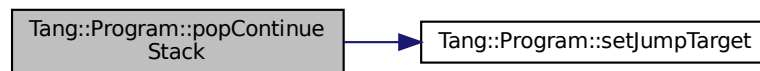
```
void Program::popContinueStack (
    size_t target )
```

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:



5.45.4.18 pushEnvironment()

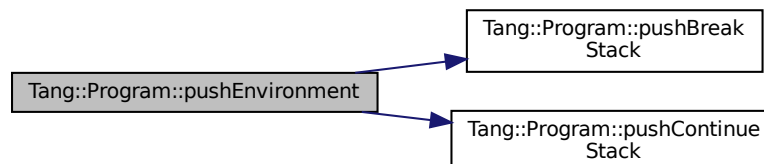
```
void Program::pushEnvironment (
    const std::shared_ptr< AstNode > & ast )
```

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:



5.45.4.19 setFunctionStackDeclaration()

```
bool Program::setFunctionStackDeclaration (
    size_t opcodeAddress,
    uinteger_t argc,
    uinteger_t targetPC )
```

Set the stack details of a function declaration.

Parameters

<i>opcodeAddress</i>	The location of the FUNCTION opcode.
<i>argc</i>	The argument count to set.
<i>targetPC</i>	The bytecode address of the start of the function.

5.45.4.20 setJumpTarget()

```
bool Program::setJumpTarget (
    size_t opcodeAddress,
    Tang::uinteger_t jumpTarget )
```

Set the target address of a Jump opcode.

Parameters

<i>opcodeAddress</i>	The location of the jump statement.
<i>jumpTarget</i>	The address to jump to.

Returns

Whether or not the jumpTarget was set.

5.45.5 Member Data Documentation

5.45.5.1 functionsDeclared

```
std::map<std::string, std::pair<uinteger_t, uinteger_t> > Tang::Program::functionsDeclared
```

Key/value pair of the function declaration information.

The key is the name of the function. The value is a pair of the *argc* value and the *targetPC* value.

The documentation for this class was generated from the following files:

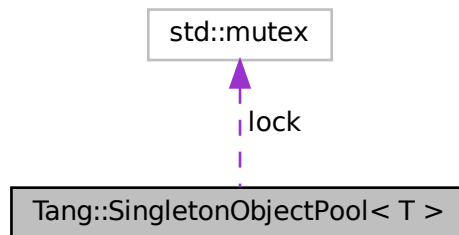
- [include/program.hpp](#)
- [src/program-dumpBytecode.cpp](#)
- [src/program-execute.cpp](#)
- [src/program.cpp](#)

5.46 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.

5.46.1 Detailed Description

```
template<class T>
class Tang::SingletonObjectPool< T >
```

A thread-safe, singleton object pool of the designated type.

5.46.2 Member Function Documentation

5.46.2.1 `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`.

5.46.2.2 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.

5.46.2.3 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.
------------	---------------------------------

5.46.3 Member Data Documentation

5.46.3.1 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.

5.46.3.2 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](#)

5.47 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](#).

5.47.1 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).

5.47.2 Constructor & Destructor Documentation

5.47.2.1 TangBase()

```
TangBase::TangBase ( )
```

The constructor.

Isn't it glorious.

5.47.3 Member Function Documentation

5.47.3.1 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:

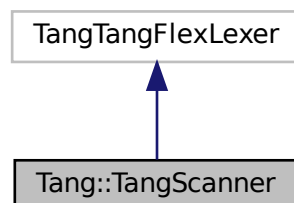
- include/[tangBase.hpp](#)
- src/[tangBase.cpp](#)

5.48 Tang::TangScanner Class Reference

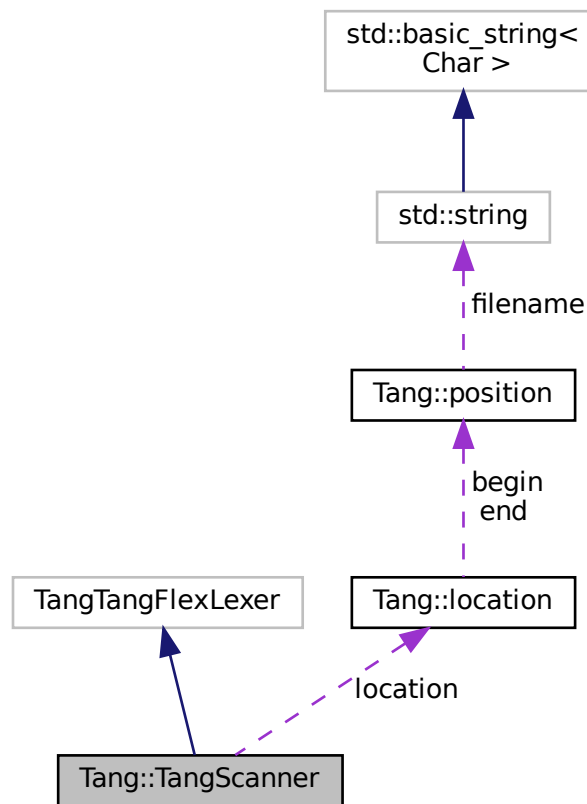
The Flex lexer class for the main Tang language.

```
#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.

5.48.1 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.

5.48.2 Constructor & Destructor Documentation

5.48.2.1 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)

5.48.3 Member Function Documentation

5.48.3.1 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:

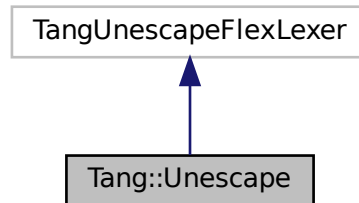
- include/[tangScanner.hpp](#)

5.49 Tang::Unescape Class Reference

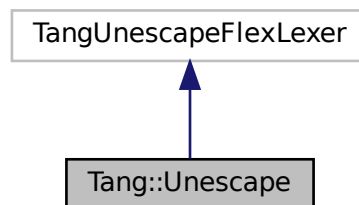
The Flex lexer class for the main Tang language.

```
#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.

5.49.1 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.

5.49.2 Constructor & Destructor Documentation

5.49.2.1 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)

5.49.3 Member Function Documentation

5.49.3.1 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:

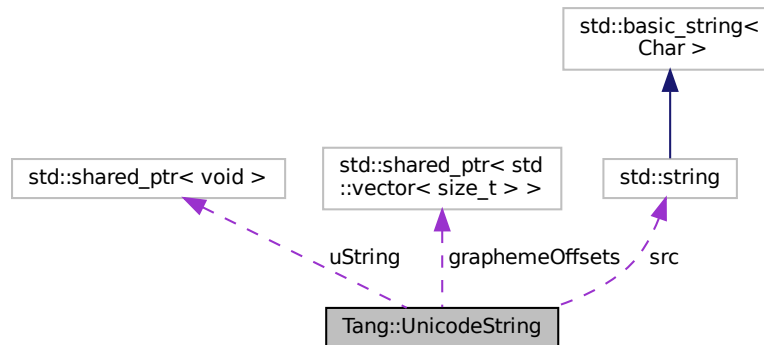
- include/[unescape.hpp](#)

5.50 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.

5.50.1 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.

5.50.2 Constructor & Destructor Documentation

5.50.2.1 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.
------------	-------------------------

5.50.3 Member Function Documentation

5.50.3.1 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.

5.50.3.2 length()

```
size_t UnicodeString::length ( ) const
```

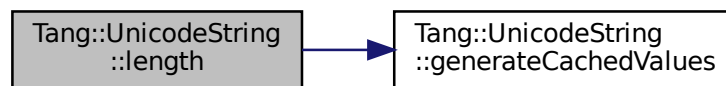
Return the length of the [UnicodeString](#) in graphemes.

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:



5.50.3.3 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](#).

5.50.3.4 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.

5.50.3.5 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.

5.50.3.6 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.

5.50.3.7 substr()

```
std::string UnicodeString::substr (
    size_t position,
    size_t length ) const
```

Return a Unicode grapheme-aware substring.

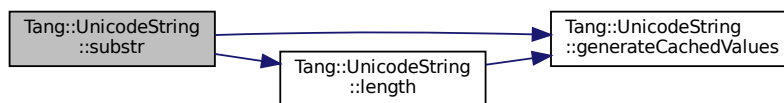
Parameters

<i>position</i>	The 0-based position of the first grapheme.
<i>length</i>	The maximum number of graphemes to return.

Returns

The requested substring.

Here is the call graph for this function:



The documentation for this class was generated from the following files:

- [include/unicodeString.hpp](#)
- [src/unicodeString.cpp](#)

Chapter 6

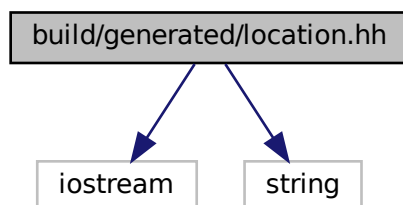
File Documentation

6.1 build/generated/location.hh File Reference

Define the Tang ::location class.

```
#include <iostream>
#include <string>
```

Include dependency graph for location.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::position](#)
A point in a source file.
- class [Tang::location](#)
Two points in a source file.

Macros

- `#define YY_NULLPTR ((void*)0)`

Functions

- position & [Tang::operator+=](#) (position &res, position::counter_type width)
Add width columns, in place.
- position [Tang::operator+](#) (position res, position::counter_type width)
Add width columns.
- position & [Tang::operator-=](#) (position &res, position::counter_type width)
Subtract width columns, in place.
- position [Tang::operator-](#) (position res, position::counter_type width)
Subtract width columns.
- template<typename YYChar >
std::basic_ostream< YYChar > & [Tang::operator<<](#) (std::basic_ostream< YYChar > &ostr, const position &pos)
Intercept output stream redirection.
- location & [Tang::operator+=](#) (location &res, const location &end)
Join two locations, in place.
- location [Tang::operator+](#) (location res, const location &end)
Join two locations.
- location & [Tang::operator+=](#) (location &res, location::counter_type width)
Add width columns to the end position, in place.
- location [Tang::operator+](#) (location res, location::counter_type width)
Add width columns to the end position.
- location & [Tang::operator-=](#) (location &res, location::counter_type width)
Subtract width columns to the end position, in place.
- location [Tang::operator-](#) (location res, location::counter_type width)
Subtract width columns to the end position.
- template<typename YYChar >
std::basic_ostream< YYChar > & [Tang::operator<<](#) (std::basic_ostream< YYChar > &ostr, const location &loc)
Intercept output stream redirection.

6.1.1 Detailed Description

Define the Tang ::location class.

6.1.2 Function Documentation

6.1.2.1 operator<<() [1/2]

```
template<typename YYChar >
std::basic_ostream<YYChar>& Tang::operator<< (
    std::basic_ostream< YYChar > & ostr,
    const location & loc )
```

Intercept output stream redirection.

Parameters

<i>ostr</i>	the destination output stream
<i>loc</i>	a reference to the location to redirect

Avoid duplicate information.

6.1.2.2 operator<<() [2/2]

```
template<typename YYChar >
std::basic_ostream<YYChar>& Tang::operator<< (
    std::basic_ostream< YYChar > & ostr,
    const position & pos )
```

Intercept output stream redirection.

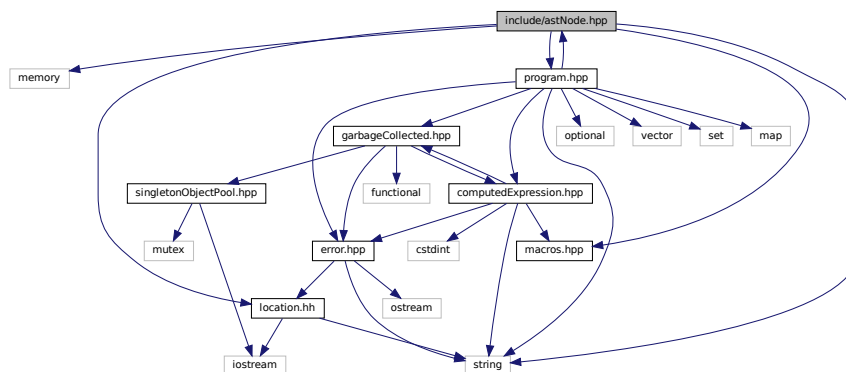
Parameters

<i>ostr</i>	the destination output stream
<i>pos</i>	a reference to the position to redirect

6.2 include/astNode.hpp File Reference

Declare the [Tang::AstNode](#) base class.

```
#include <memory>
#include <string>
#include "location.hh"
#include "macros.hpp"
#include "program.hpp"
Include dependency graph for astNode.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNode](#)
Base class for representing nodes of an Abstract Syntax Tree (AST).

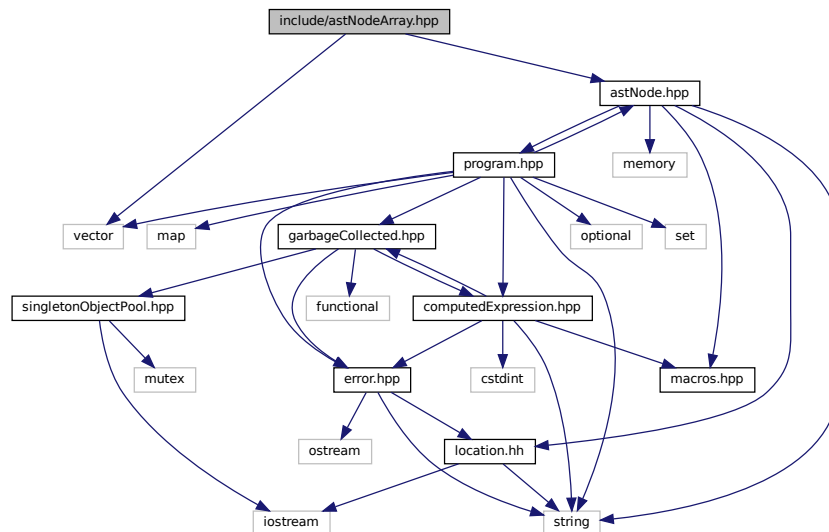
6.2.1 Detailed Description

Declare the [Tang::AstNode](#) base class.

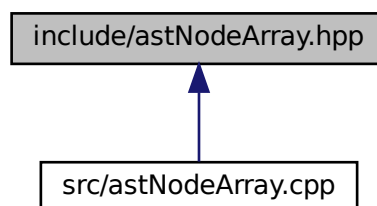
6.3 include/astNodeArray.hpp File Reference

Declare the [Tang::AstNodeArray](#) class.

```
#include <vector>
#include "astNode.hpp"
Include dependency graph for astNodeArray.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeArray](#)
An *AstNode* that represents an array literal.

6.3.1 Detailed Description

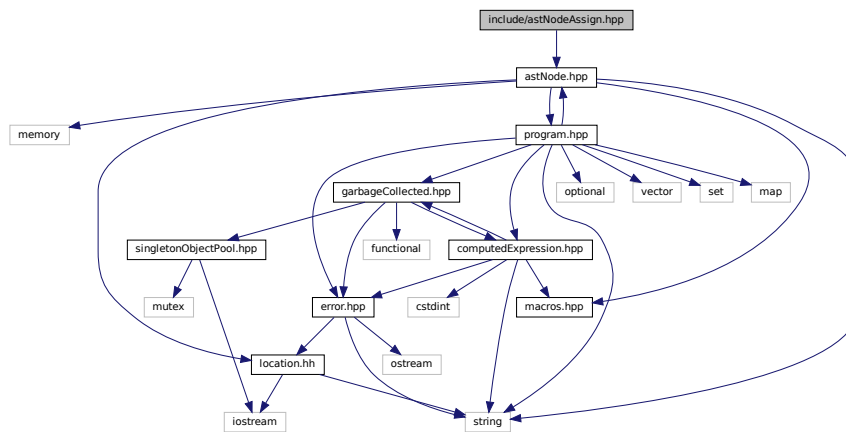
Declare the [Tang::AstNodeArray](#) class.

6.4 include/astNodeAssign.hpp File Reference

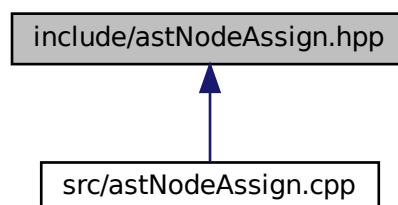
Declare the [Tang::AstNodeAssign](#) class.

```
#include "astNode.hpp"
```

Include dependency graph for astNodeAssign.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeAssign](#)
An *AstNode* that represents a binary expression.

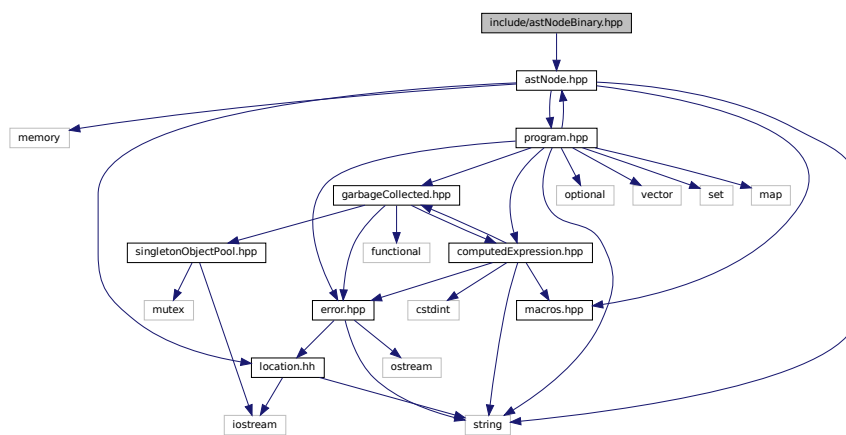
6.4.1 Detailed Description

Declare the [Tang::AstNodeAssign](#) class.

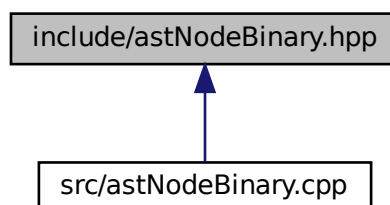
6.5 include/astNodeBinary.hpp File Reference

Declare the [Tang::AstNodeBinary](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeBinary.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeBinary](#)
An [AstNode](#) that represents a binary expression.

6.5.1 Detailed Description

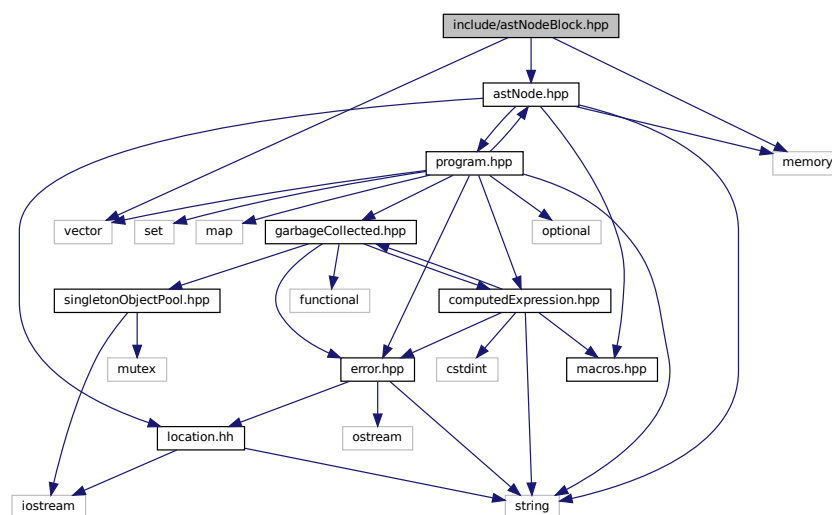
Declare the [Tang::AstNodeBinary](#) class.

6.6 include/astNodeBlock.hpp File Reference

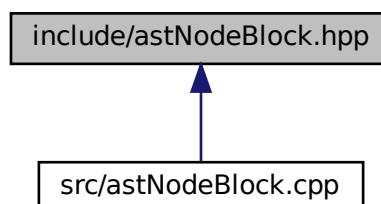
Declare the [Tang::AstNodeBlock](#) class.

```
#include <vector>
#include <memory>
#include "astNode.hpp"
```

Include dependency graph for astNodeBlock.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeBlock](#)
An [AstNode](#) that represents a code block.

6.6.1 Detailed Description

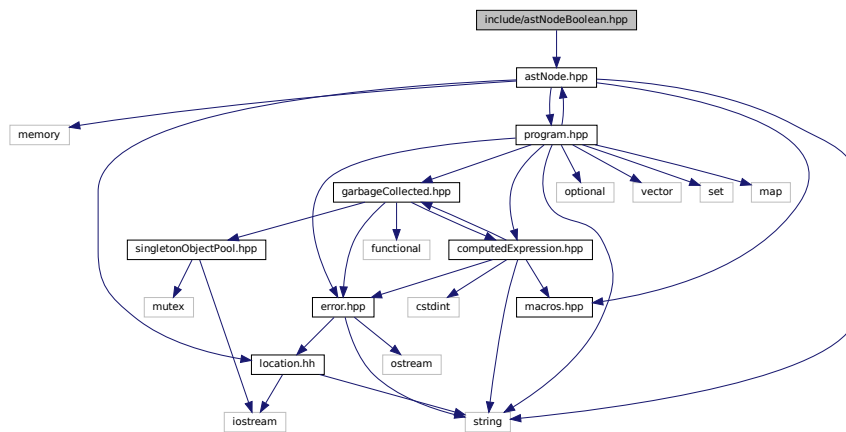
Declare the [Tang::AstNodeBlock](#) class.

6.7 include/astNodeBoolean.hpp File Reference

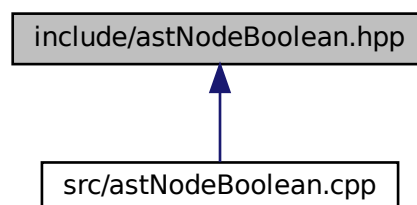
Declare the [Tang::AstNodeBoolean](#) class.

```
#include "astNode.hpp"
```

Include dependency graph for astNodeBoolean.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class `Tang::AstNodeBoolean`
An `AstNode` that represents a boolean literal.

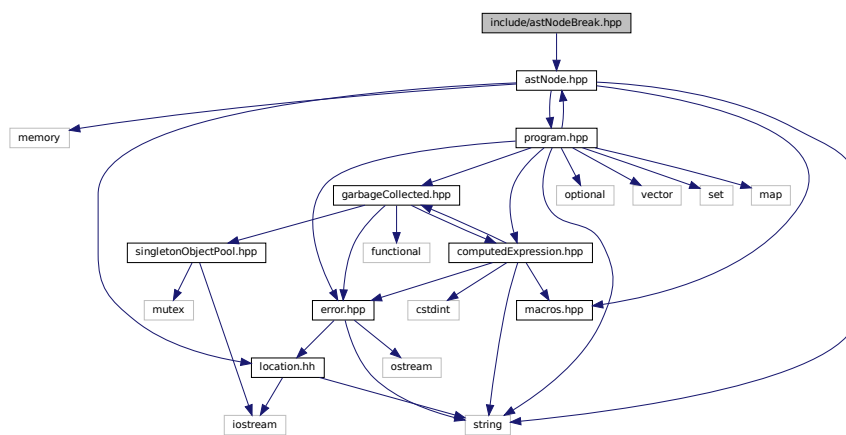
6.7.1 Detailed Description

Declare the `Tang::AstNodeBoolean` class.

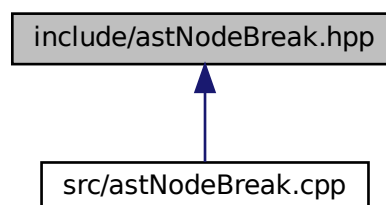
6.8 include/astNodeBreak.hpp File Reference

Declare the `Tang::AstNodeBreak` class.

```
#include "astNode.hpp"
Include dependency graph for astNodeBreak.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeBreak](#)
An *AstNode* that represents a *break* statement.

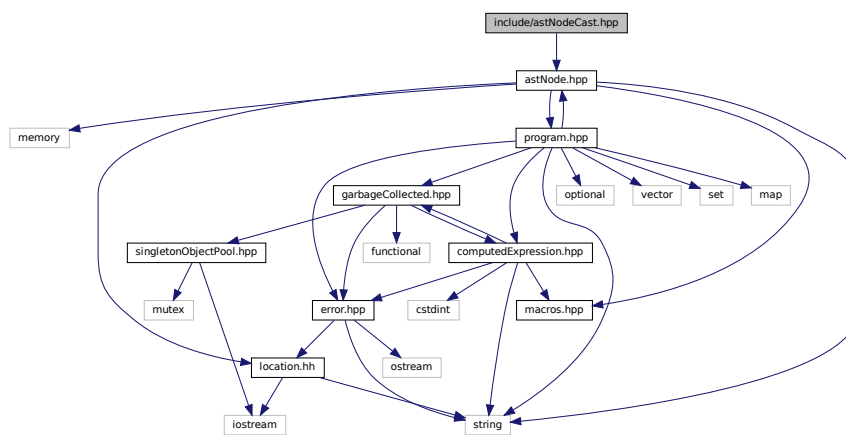
6.8.1 Detailed Description

Declare the [Tang::AstNodeBreak](#) class.

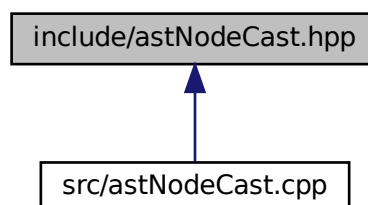
6.9 include/astNodeCast.hpp File Reference

Declare the [Tang::AstNodeCast](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeCast.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeCast](#)
An *AstNode* that represents a typecast of an expression.

6.9.1 Detailed Description

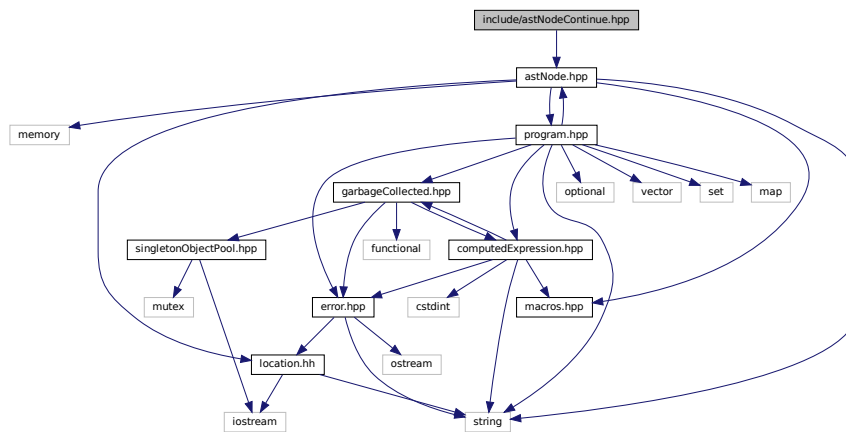
Declare the [Tang::AstNodeCast](#) class.

6.10 include/astNodeContinue.hpp File Reference

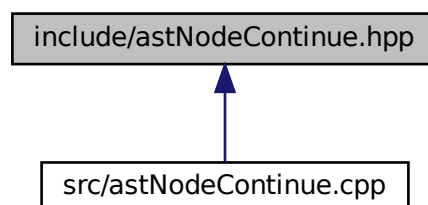
Declare the [Tang::AstNodeContinue](#) class.

```
#include "astNode.hpp"
```

Include dependency graph for astNodeContinue.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeContinue](#)

An [AstNode](#) that represents a *continue* statement.

6.10.1 Detailed Description

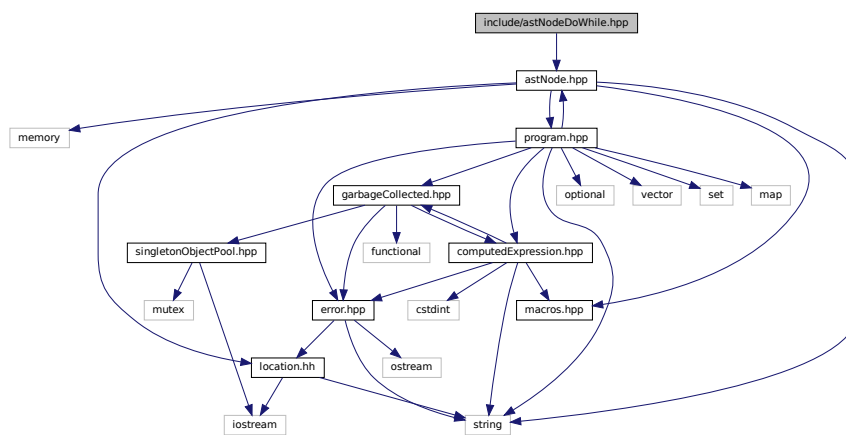
Declare the [Tang::AstNodeContinue](#) class.

6.11 include/astNodeDoWhile.hpp File Reference

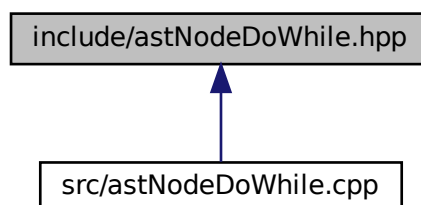
Declare the [Tang::AstNodeDoWhile](#) class.

```
#include "astNode.hpp"
```

Include dependency graph for astNodeDoWhile.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeDoWhile](#)
An *AstNode* that represents a do..while statement.

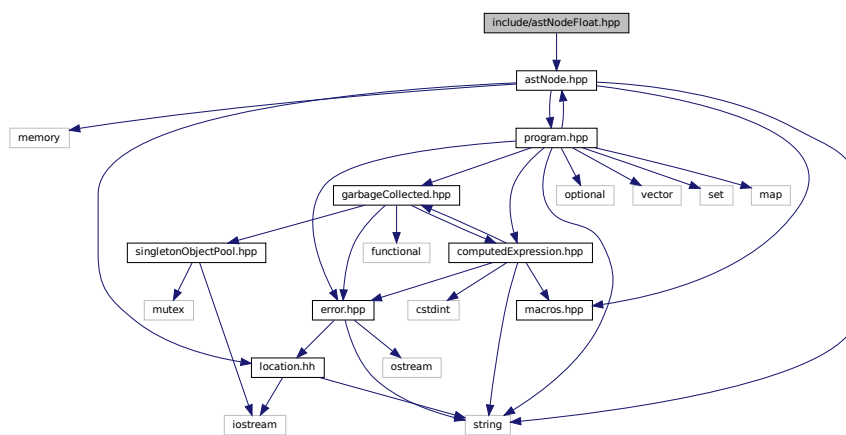
6.11.1 Detailed Description

Declare the [Tang::AstNodeDoWhile](#) class.

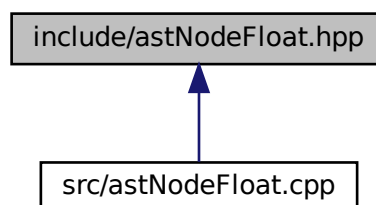
6.12 include/astNodeFloat.hpp File Reference

Declare the [Tang::AstNodeFloat](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeFloat.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeFloat](#)
An *AstNode* that represents an float literal.

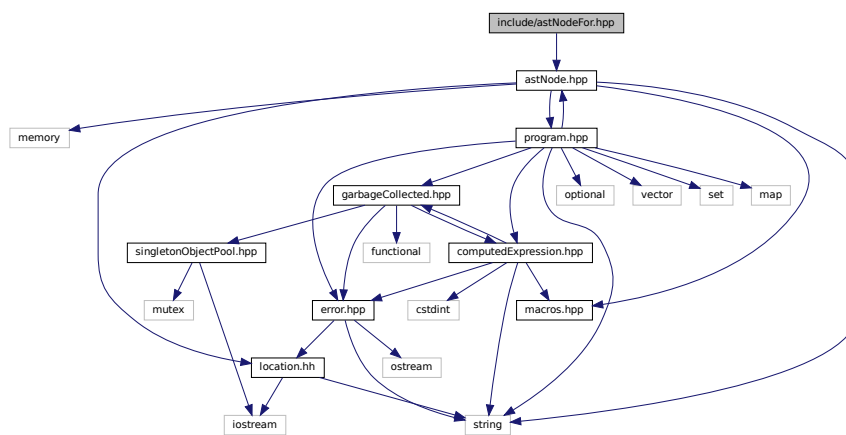
6.12.1 Detailed Description

Declare the [Tang::AstNodeFloat](#) class.

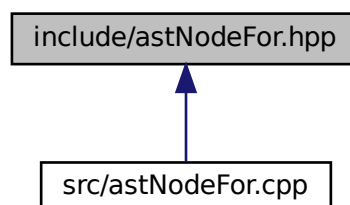
6.13 include/astNodeFor.hpp File Reference

Declare the [Tang::AstNodeFor](#) class.

```
#include "astNode.hpp"
Include dependency graph for astNodeFor.hpp:
```



This graph shows which files directly or indirectly include this file:

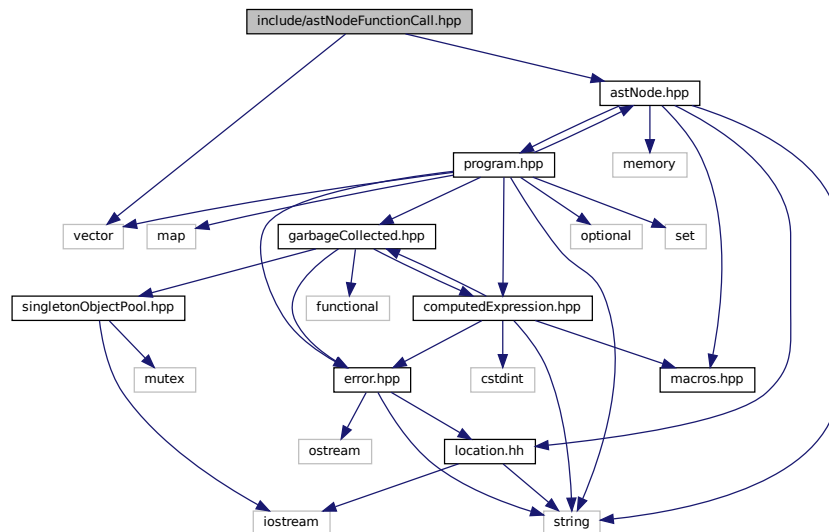


- class `Tang::AstNodeFor`
An `AstNode` that represents an `if()` statement.

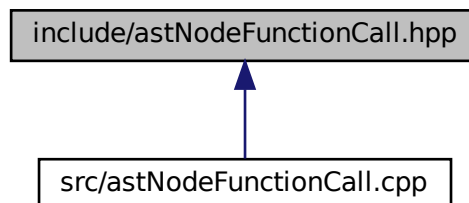
Declare the `Tang::AstNodeFor` class.

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.

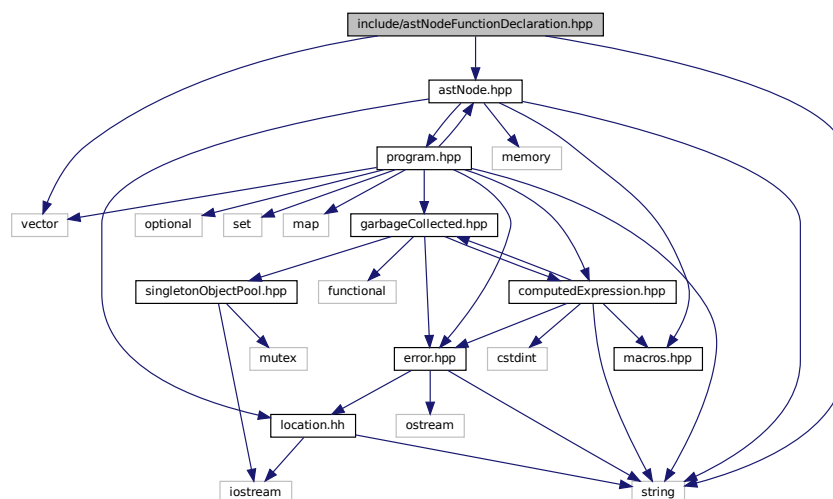
6.14.1 Detailed Description

Declare the [Tang::AstNodeFunctionCall](#) class.

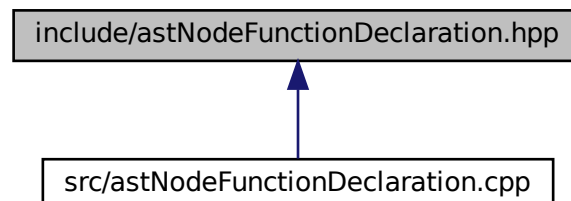
6.15 include/astNodeFunctionDeclaration.hpp File Reference

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.

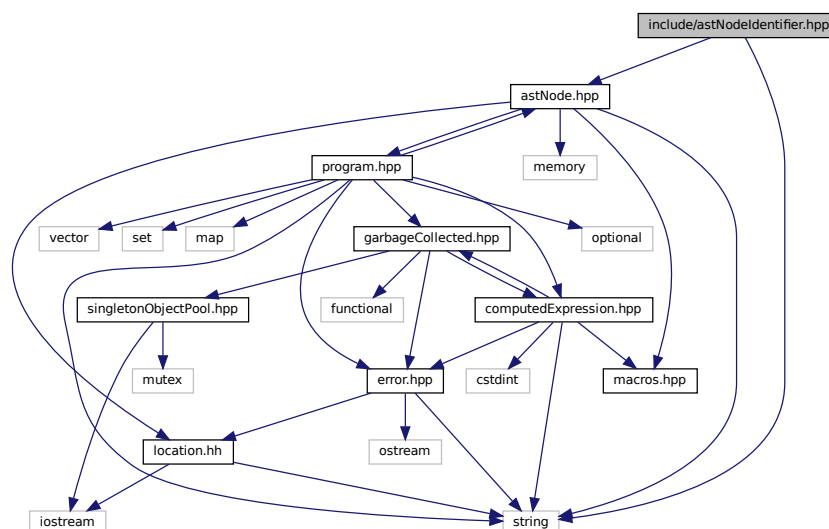
6.15.1 Detailed Description

Declare the [Tang::AstNodeFunctionDeclaration](#) class.

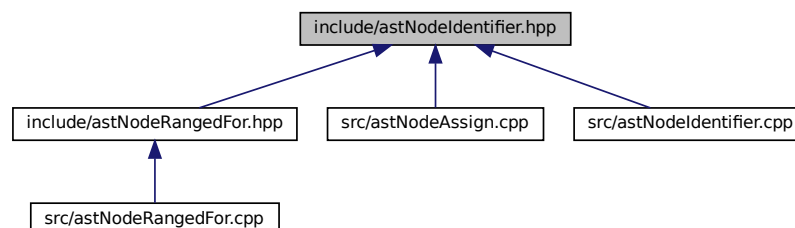
6.16 include/astNodeIdentifier.hpp File Reference

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.

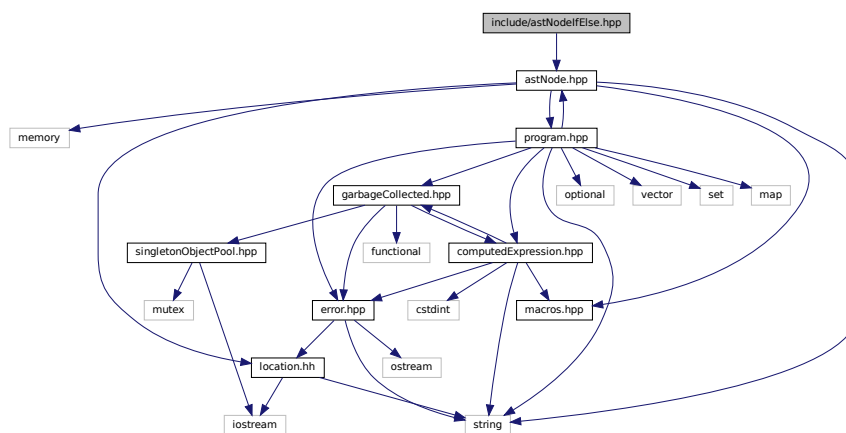
6.16.1 Detailed Description

Declare the [Tang::AstNodeIdentifier](#) class.

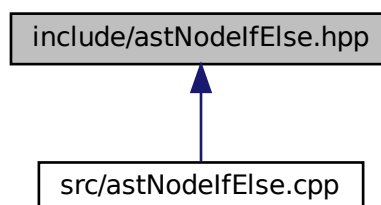
6.17 include/astNodeIfElse.hpp File Reference

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.

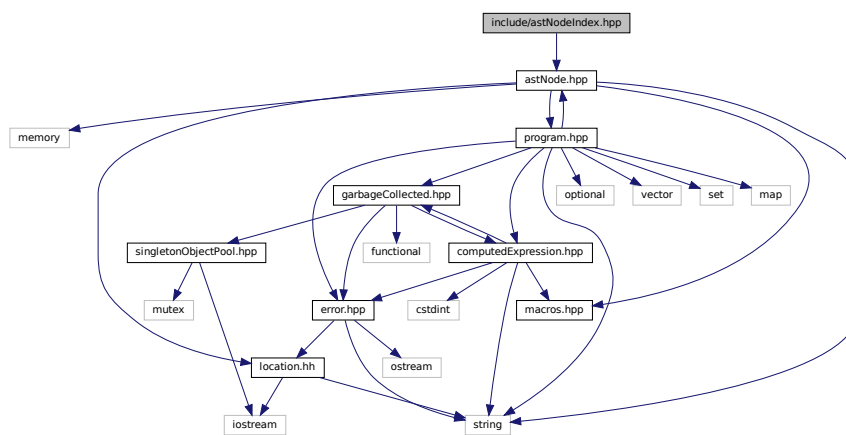
6.17.1 Detailed Description

Declare the [Tang::AstNodeIfElse](#) class.

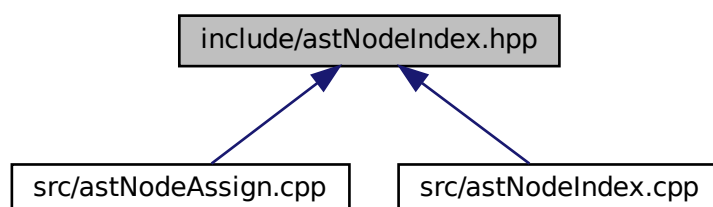
6.18 include/astNodeIndex.hpp File Reference

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

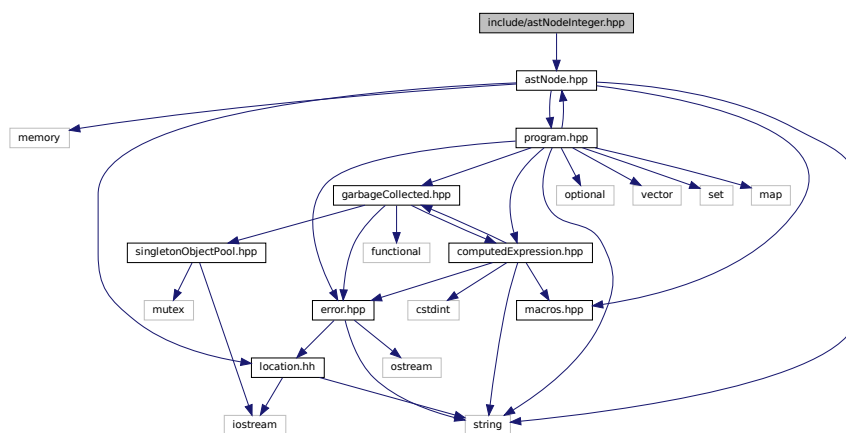
Declare the [Tang::AstNodeIndex](#) class.

6.19 include/astNodeInteger.hpp File Reference

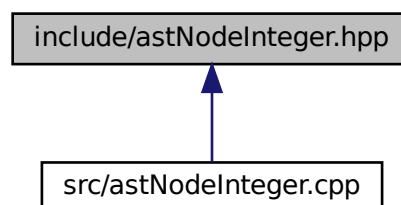
Declare the [Tang::AstNodeInteger](#) class.

```
#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

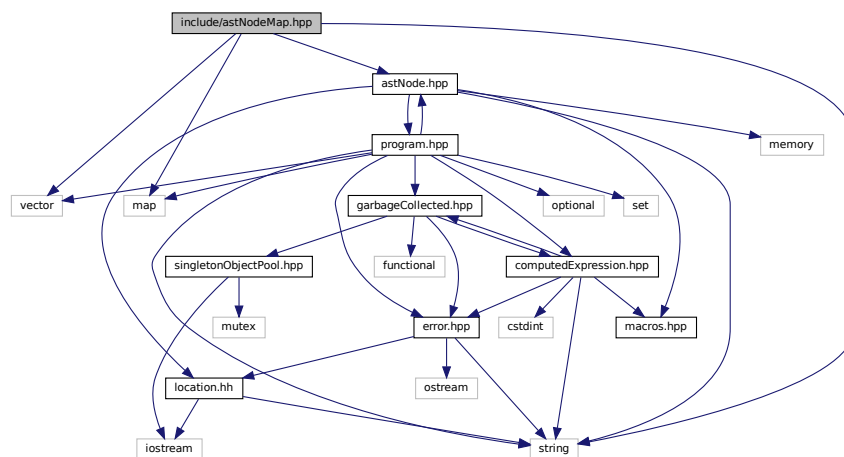
Declare the [Tang::AstNodeInteger](#) class.

6.20 include/astNodeMap.hpp File Reference

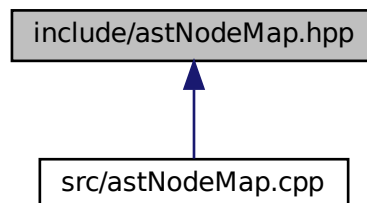
Declare the [Tang::AstNodeMap](#) class.

```
#include <vector>
#include <map>
#include <string>
#include "astNode.hpp"
```

Include dependency graph for astNodeMap.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::AstNodeMap](#)
An *AstNode* that represents a map literal.

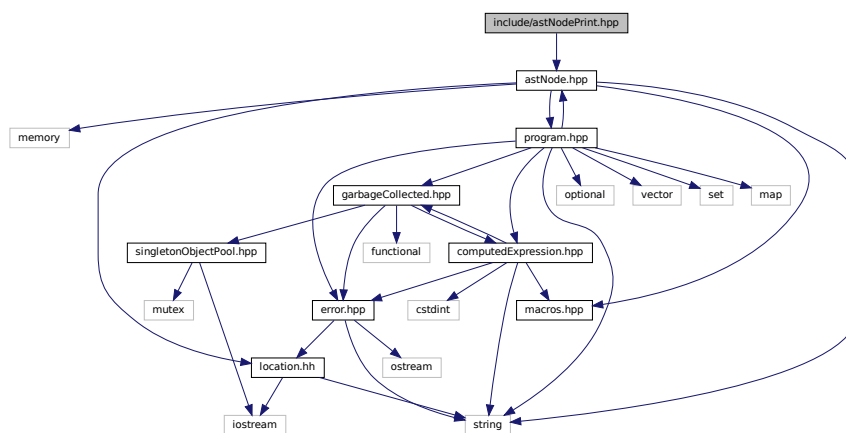
6.20.1 Detailed Description

Declare the [Tang::AstNodeMap](#) class.

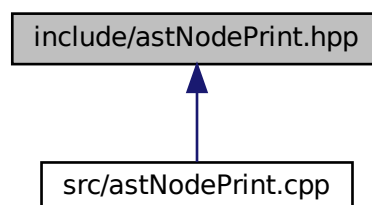
6.21 include/astNodePrint.hpp File Reference

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.

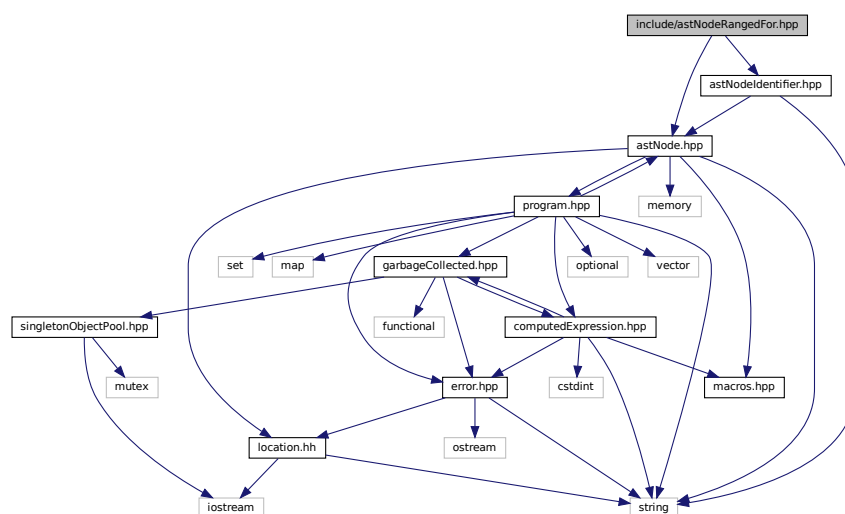
6.21.1 Detailed Description

Declare the [Tang::AstNodePrint](#) class.

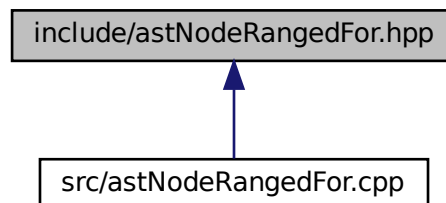
6.22 include/astNodeRangedFor.hpp File Reference

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.

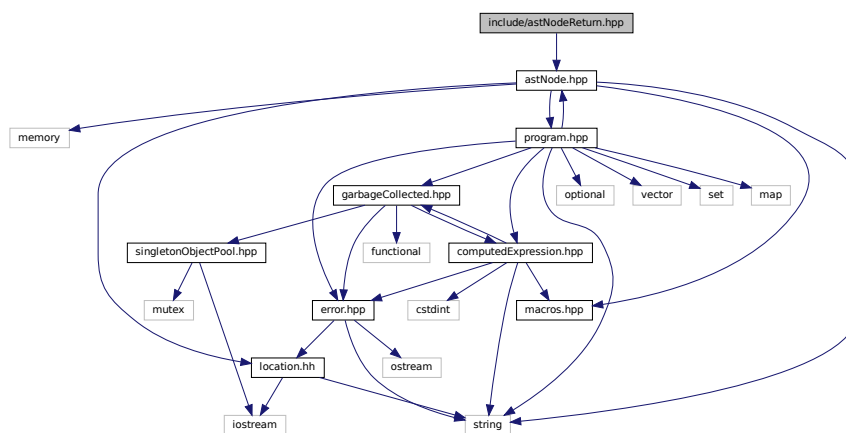
6.22.1 Detailed Description

Declare the [Tang::AstNodeRangedFor](#) class.

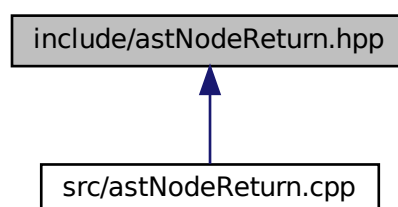
6.23 include/astNodeReturn.hpp File Reference

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.

6.23.1 Detailed Description

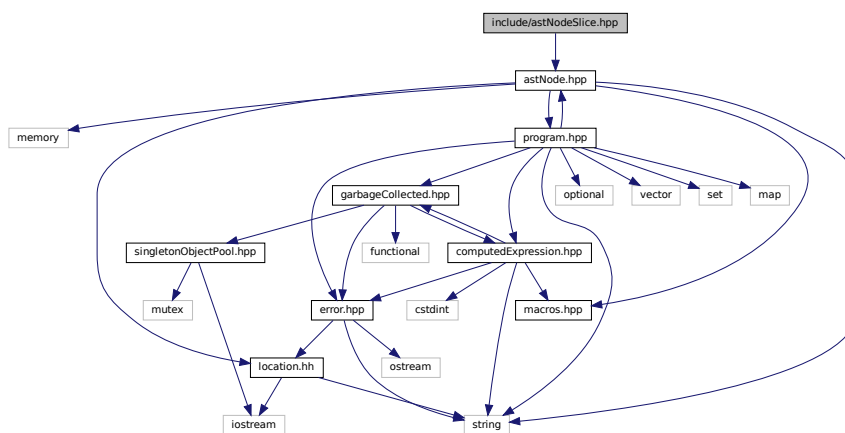
Declare the `Tang::AstNodeReturn` class.

6.24 include/astNodeSlice.hpp File Reference

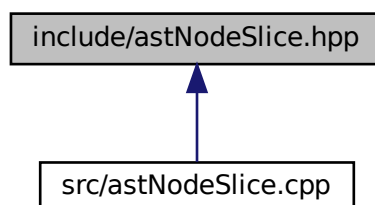
Declare the `Tang::AstNodeSlice` class.

```
#include "astNode.hpp"
```

Include dependency graph for astNodeSlice.hpp:



This graph shows which files directly or indirectly include this file:



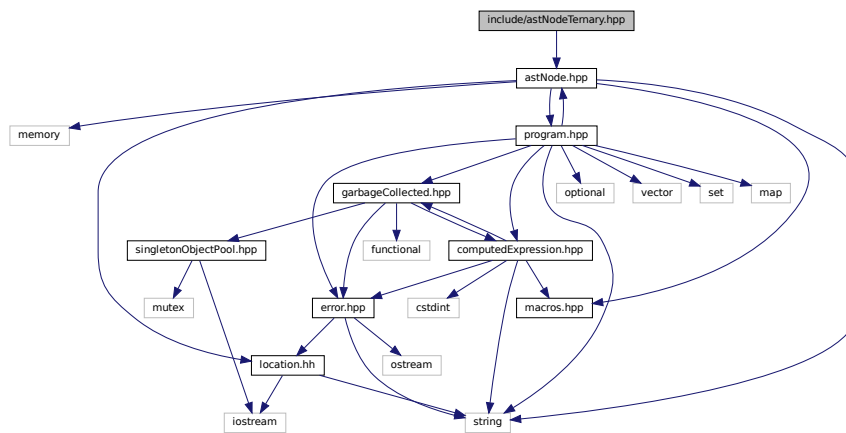
6.25.1 Detailed Description

Declare the [Tang::AstNodeString](#) class.

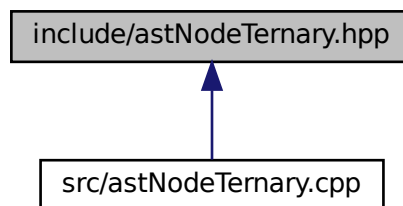
6.26 include/astNodeTernary.hpp File Reference

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.

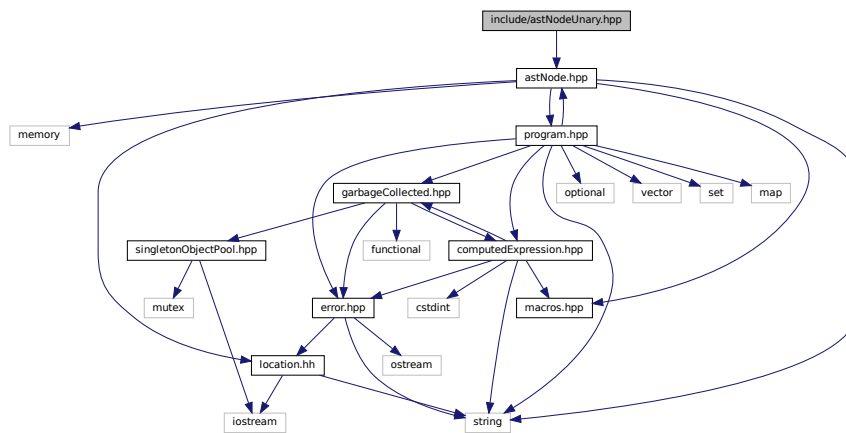
6.26.1 Detailed Description

Declare the [Tang::AstNodeTernary](#) class.

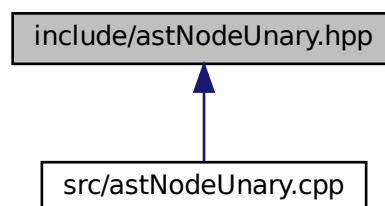
6.27 include/astNodeUnary.hpp File Reference

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.

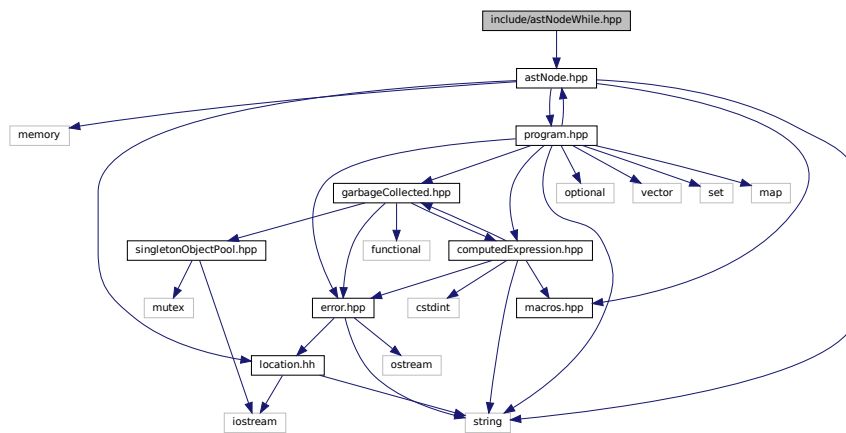
6.27.1 Detailed Description

Declare the [Tang::AstNodeUnary](#) class.

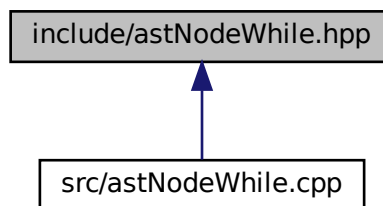
6.28 include/astNodeWhile.hpp File Reference

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.

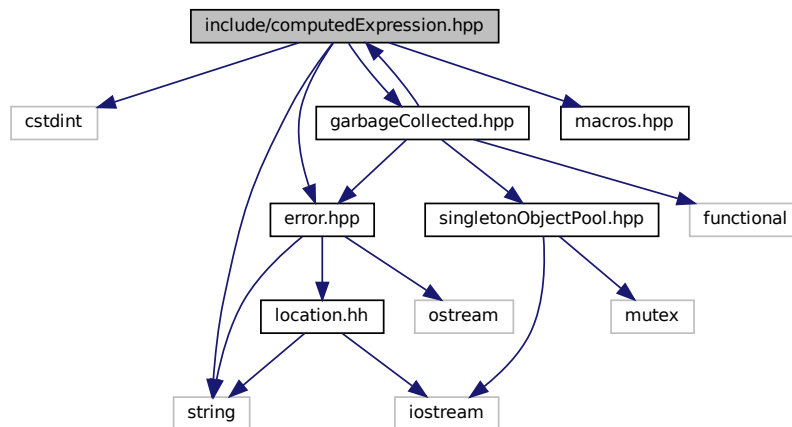
6.28.1 Detailed Description

Declare the [Tang::AstNodeWhile](#) class.

6.29 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.

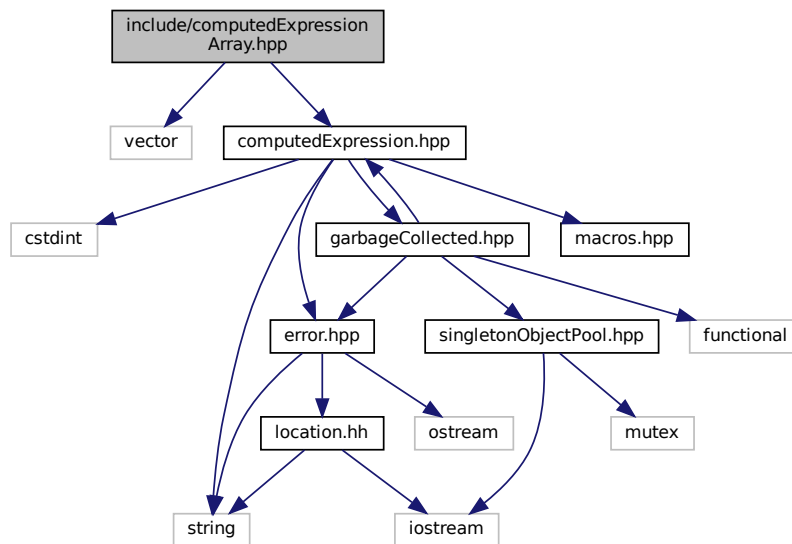
6.29.1 Detailed Description

Declare the [Tang::ComputedExpression](#) base class.

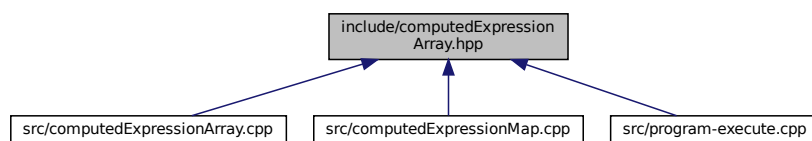
6.30 include/computedExpressionArray.hpp File Reference

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.

6.30.1 Detailed Description

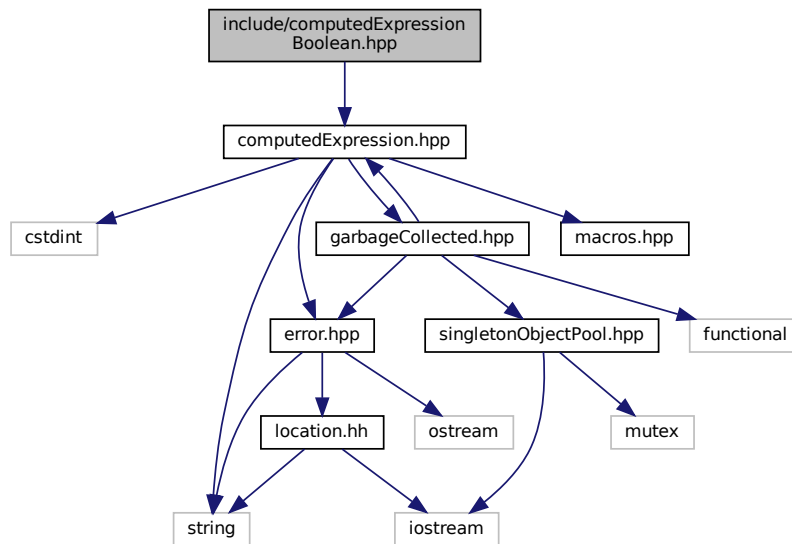
Declare the [Tang::ComputedExpressionArray](#) class.

6.31 include/computedExpressionBoolean.hpp File Reference

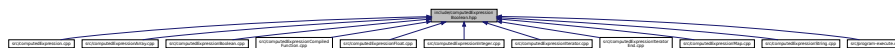
Declare the [Tang::ComputedExpressionBoolean](#) class.

```
#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.

6.31.1 Detailed Description

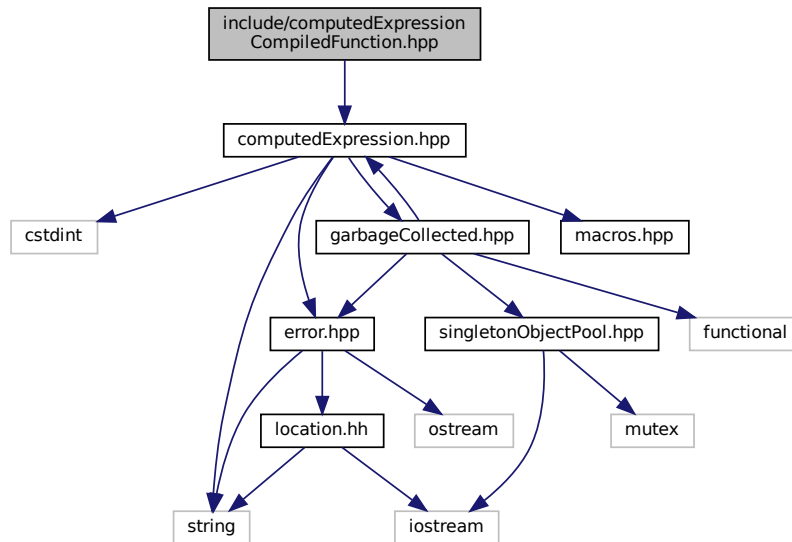
Declare the [Tang::ComputedExpressionBoolean](#) class.

6.32 include/computedExpressionCompiledFunction.hpp File Reference

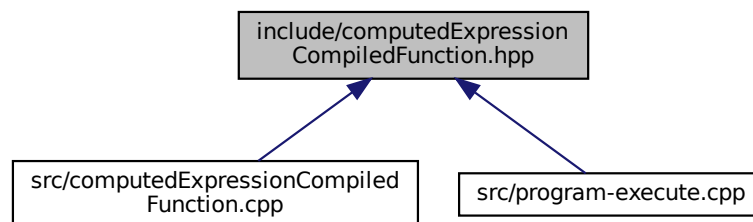
Declare the [Tang::ComputedExpressionCompiledFunction](#) class.

```
#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.

6.32.1 Detailed Description

Declare the [Tang::ComputedExpressionCompiledFunction](#) class.

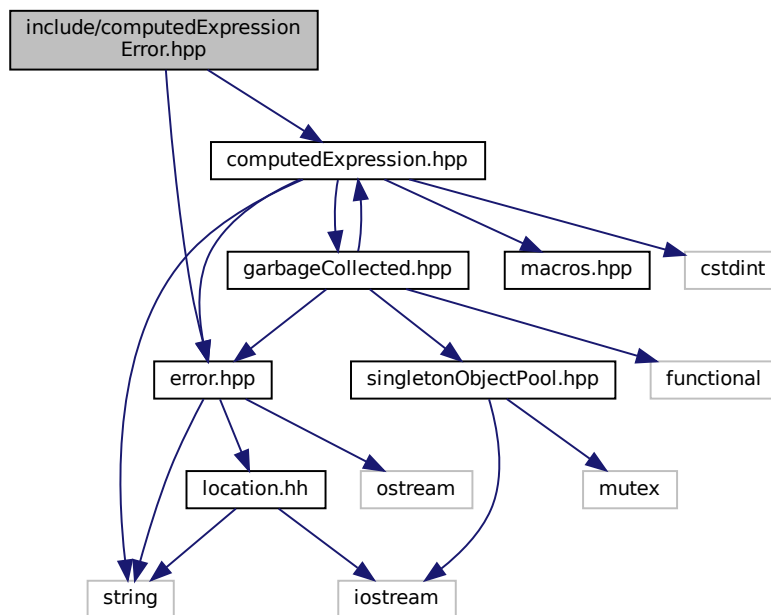
6.33 include/computedExpressionError.hpp File Reference

Declare the [Tang::ComputedExpressionError](#) class.

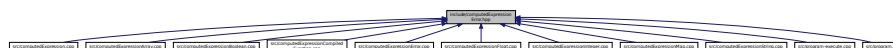
```
#include "computedExpression.hpp"
```

```
#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](#).

6.33.1 Detailed Description

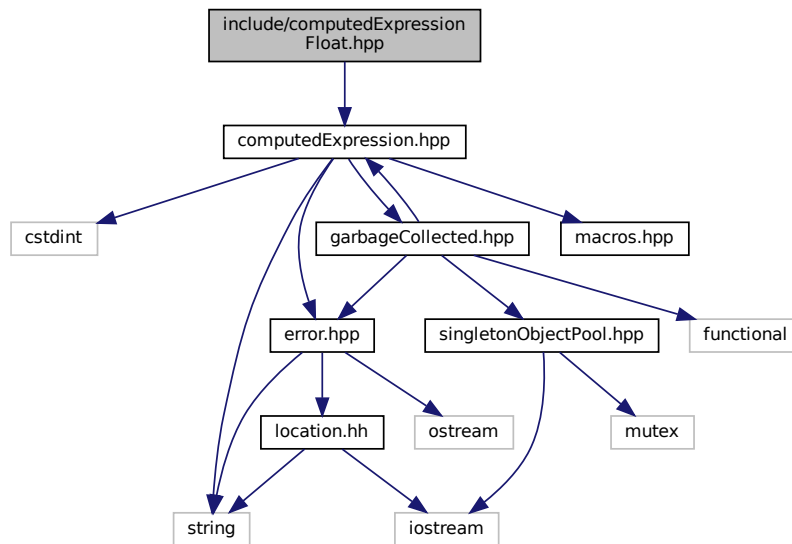
Declare the [Tang::ComputedExpressionError](#) class.

6.34 include/computedExpressionFloat.hpp File Reference

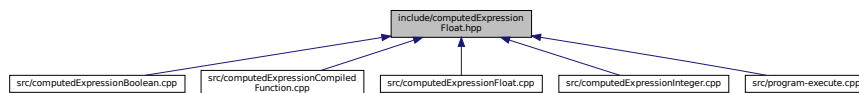
Declare the [Tang::ComputedExpressionFloat](#) class.

```
#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.

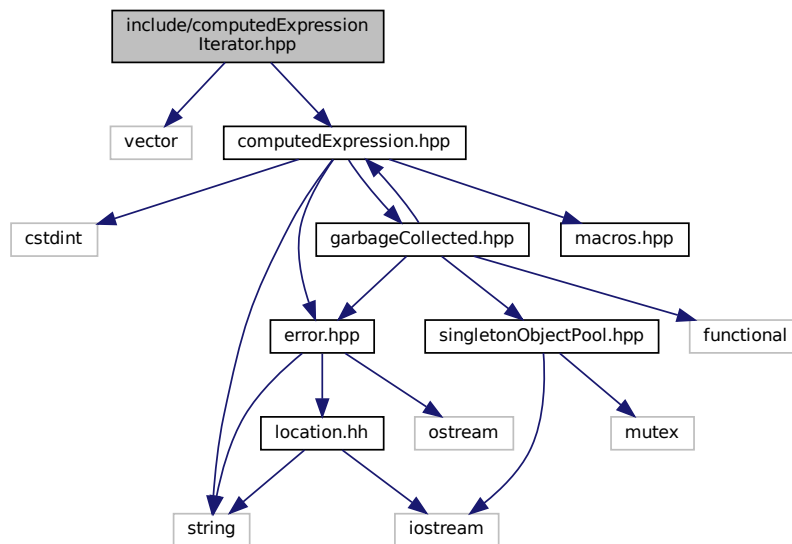
6.34.1 Detailed Description

Declare the [Tang::ComputedExpressionFloat](#) class.

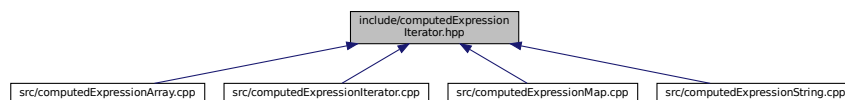
6.36 include/computedExpressionIterator.hpp File Reference

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.

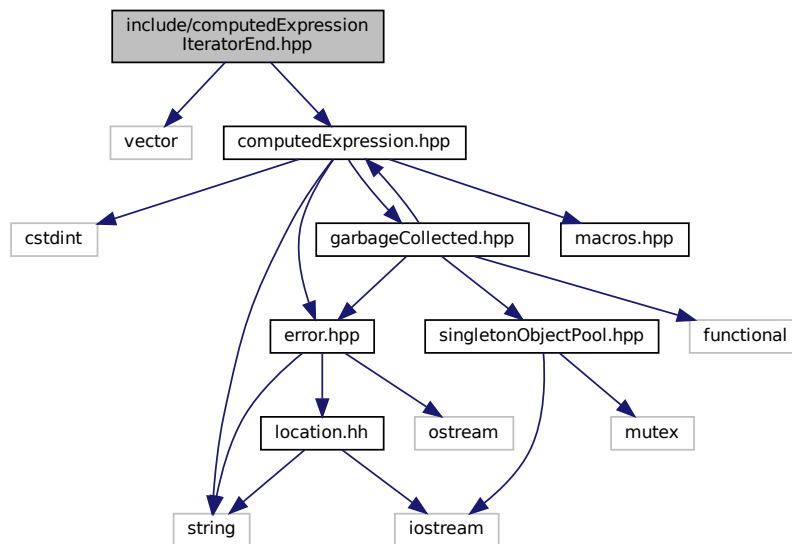
6.36.1 Detailed Description

Declare the [Tang::ComputedExpressionIterator](#) class.

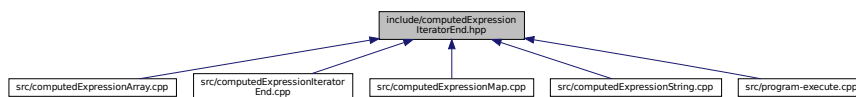
6.37 include/computedExpressionIteratorEnd.hpp File Reference

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.

6.37.1 Detailed Description

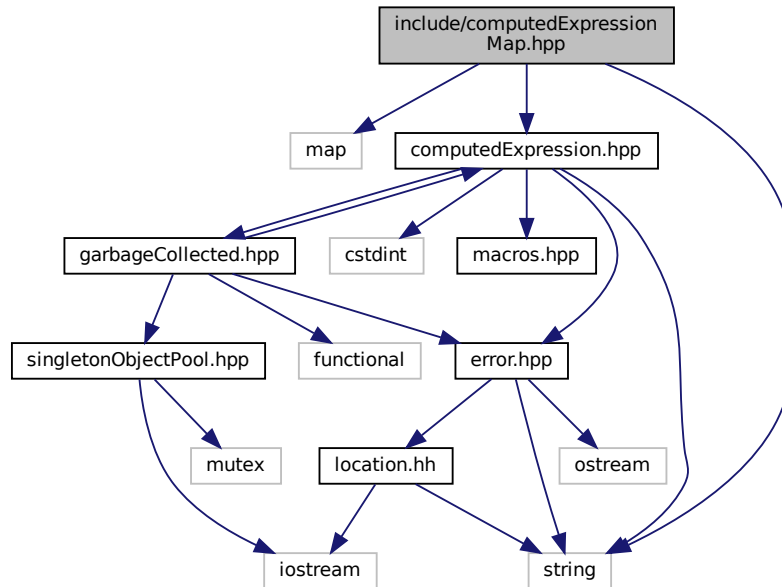
Declare the [Tang::ComputedExpressionIteratorEnd](#) class.

6.38 include/computedExpressionMap.hpp File Reference

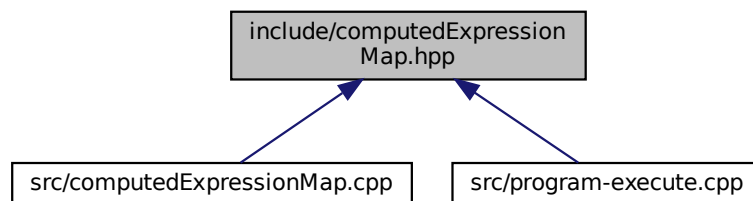
Declare the [Tang::ComputedExpressionMap](#) class.

```
#include <map>
#include <string>
#include "computedExpression.hpp"
```

Include dependency graph for computedExpressionMap.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Tang::ComputedExpressionMap](#)
Represents an Map that is the result of a computation.

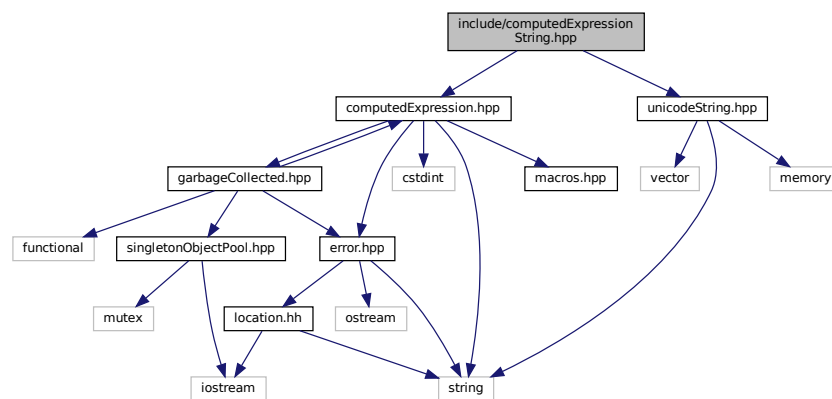
6.38.1 Detailed Description

Declare the [Tang::ComputedExpressionMap](#) class.

6.39 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.

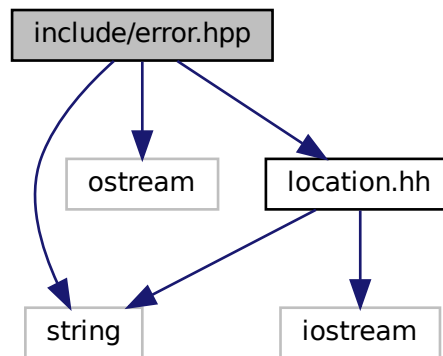
6.39.1 Detailed Description

Declare the [Tang::ComputedExpressionString](#) class.

6.40 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.

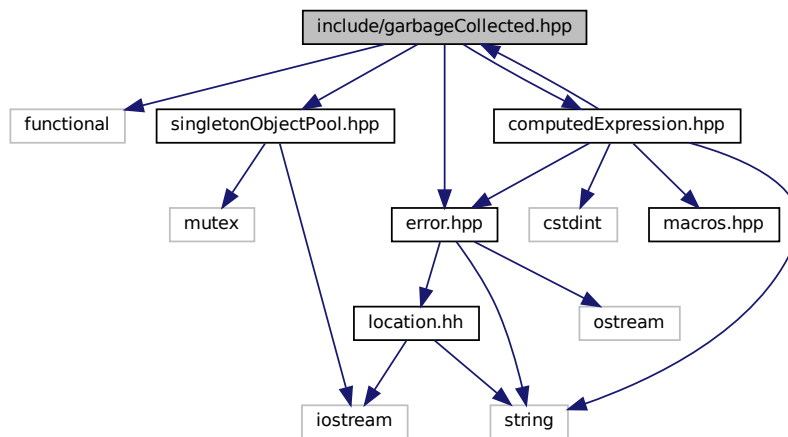
6.40.1 Detailed Description

Declare the [Tang::Error](#) class used to describe syntax and runtime errors.

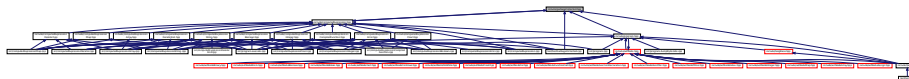
6.41 include/garbageCollected.hpp File Reference

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.

6.41.1 Detailed Description

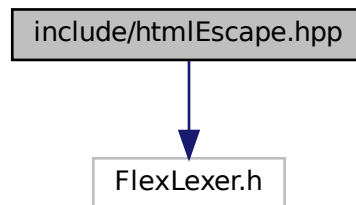
Declare the [Tang::GarbageCollected](#) class.

6.42 include/htmlEscape.hpp File Reference

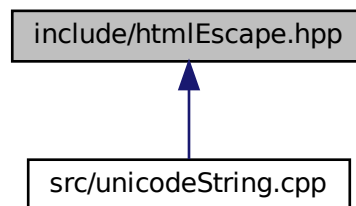
Declare the [Tang::HtmlEscape](#) used to tokenize a Tang script.

```
#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\(\)](#)

6.42.1 Detailed Description

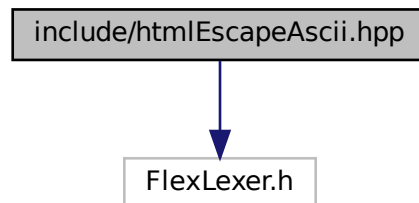
Declare the [Tang::HtmlEscape](#) used to tokenize a Tang script.

6.43 include/htmlEscapeAscii.hpp File Reference

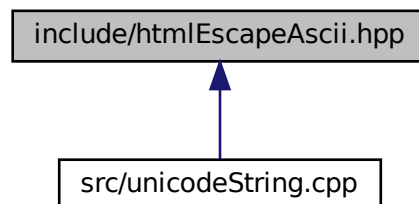
Declare the [Tang::HtmlEscapeAscii](#) used to tokenize a Tang script.

```
#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\(\)](#)

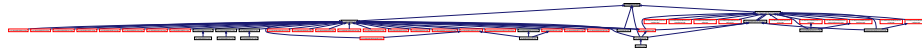
6.43.1 Detailed Description

Declare the [Tang::HtmlEscapeAscii](#) used to tokenize a Tang script.

6.44 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.

6.44.1 Detailed Description

Contains generic macros.

6.45 include/opcode.hpp File Reference

Declare the Opcodes used in the Bytecode representation of a program.

This graph shows which files directly or indirectly include this file:



Enumerations

- enum class `Tang::Opcode` {
`POP` , `PEEK` , `POKE` , `COPY` ,
`JMP` , `JMPF` , `JMPF_POP` , `JMPT` ,
`JMPT_POP` , `NULLVAL` , `INTEGER` , `FLOAT` ,
`BOOLEAN` , `STRING` , `ARRAY` , `MAP` ,
`FUNCTION` , `ASSIGNINDEX` , `ADD` , `SUBTRACT` ,
`MULTIPLY` , `DIVIDE` , `MODULO` , `NEGATIVE` ,
`NOT` , `LT` , `LTE` , `GT` ,
`GTE` , `EQ` , `NEQ` , `INDEX` ,
`SLICE` , `GETITERATOR` , `ITERATORNEXT` , `ISITERATOREND` ,
`CASTINTEGER` , `CASTFLOAT` , `CASTBOOLEAN` , `CASTSTRING` ,
`CALLFUNC` , `RETURN` , `PRINT` }

6.45.1 Detailed Description

Declare the Opcodes used in the Bytecode representation of a program.

6.45.2 Enumeration Type Documentation

6.45.2.1 Opcode

```
enum Tang::Opcode [strong]
```

Enumerator

POP	Pop a val.
PEEK	Stack # (from fp): push val from stack #.
POKE	Stack # (from fp): Copy a val, store @ stack #.
COPY	Stack # (from fp): Deep copy val @ stack #, store @ stack #.
JMP	PC #: set pc to PC #.
JMPF	PC #: read val, if false, set pc to PC #.
JMPF_POP	PC #: pop val, if false, set pc to PC #.
JMPT	PC #: read val, if true, set pc to PC #.
JMPT_POP	PC #: pop val, if true, set pc to PC #.
NULLVAL	Push a null onto the stack.
INTEGER	Push an integer onto the stack.
FLOAT	Push a floating point number onto the stack.
BOOLEAN	Push a boolean onto the stack.
STRING	Get len, char string: push string.
ARRAY	Get len, pop len items, putting them into an array with the last array item popped first.
MAP	Get len, pop len value then key pairs, putting them into a map.
FUNCTION	Get argc, PC#: push function(argc, PC #)
ASSIGNINDEX	Pop index, pop collection, pop value, push (collection[index] = value)
ADD	Pop rhs, pop lhs, push lhs + rhs.
SUBTRACT	Pop rhs, pop lhs, push lhs - rhs.
MULTIPLY	Pop rhs, pop lhs, push lhs * rhs.
DIVIDE	Pop rhs, pop lhs, push lhs / rhs.
MODULO	Pop rhs, pop lhs, push lhs % rhs.
NEGATIVE	Pop val, push negative val.
NOT	Pop val, push logical not of val.
LT	Pop rhs, pop lhs, push lhs < rhs.
LTE	Pop rhs, pop lhs, push lhs <= rhs.
GT	Pop rhs, pop lhs, push lhs > rhs.
GTE	Pop rhs, pop lhs, push lhs >= rhs.
EQ	Pop rhs, pop lhs, push lhs == rhs.
NEQ	Pop rhs, pop lhs, push lhs != rhs.
INDEX	Pop index, pop collection, push collection[index].
SLICE	Pop skip, pop end, pop begin, pop collection, push collection[begin:end:skip].

Enumerator

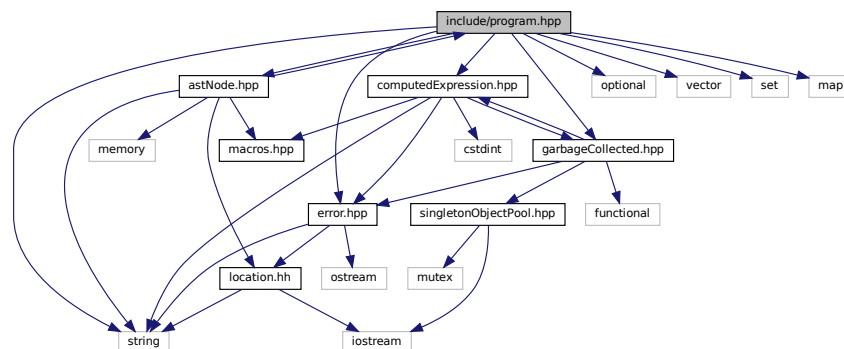
GETITERATOR	Pop a collection, push the collection iterator.
ITERATORNEXT	Pop an iterator, push the next iterator value.
ISITERATOREND	Pop a val, push bool(is val == iterator end)
CASTINTEGER	Pop a val, typecast to int, push.
CASTFLOAT	Pop a val, typecast to float, push.
CASTBOOLEAN	Pop a val, typecast to boolean, push.
CASTSTRING	Pop a val, typecast to string, push.
CALLFUNC	Get argc, Pop a function, execute function if argc matches.
RETURN	Get stack #, pop return val, pop (stack #) times, push val, restore fp, restore pc.
PRINT	Pop val, print(val), push error or NULL.

6.46 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.

6.46.1 Detailed Description

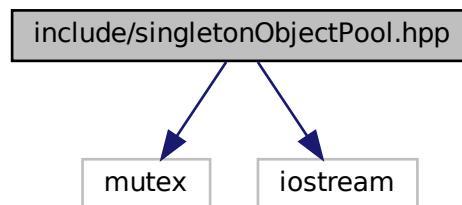
Declare the [Tang::Program](#) class used to compile and execute source code.

6.47 include/singletonObjectPool.hpp File Reference

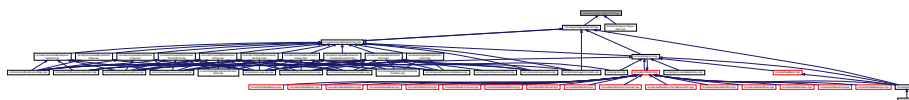
Declare the [Tang::SingletonObjectPool](#) class.

```
#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.

6.47.1 Detailed Description

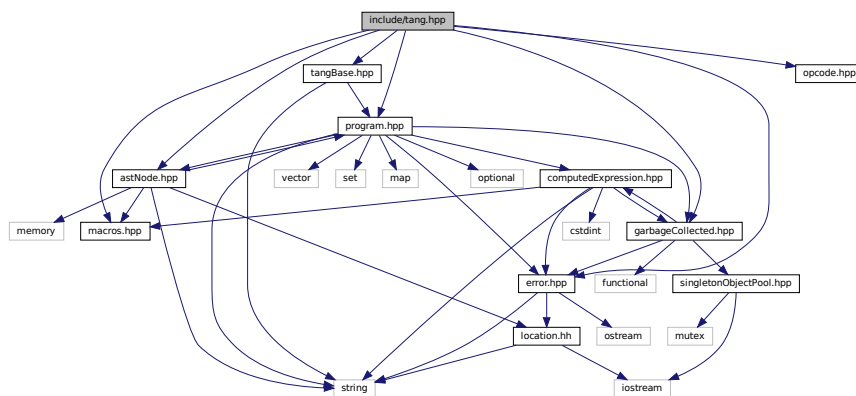
Declare the [Tang::SingletonObjectPool](#) class.

6.48 include/tang.hpp File Reference

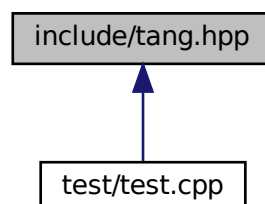
Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

```
#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:



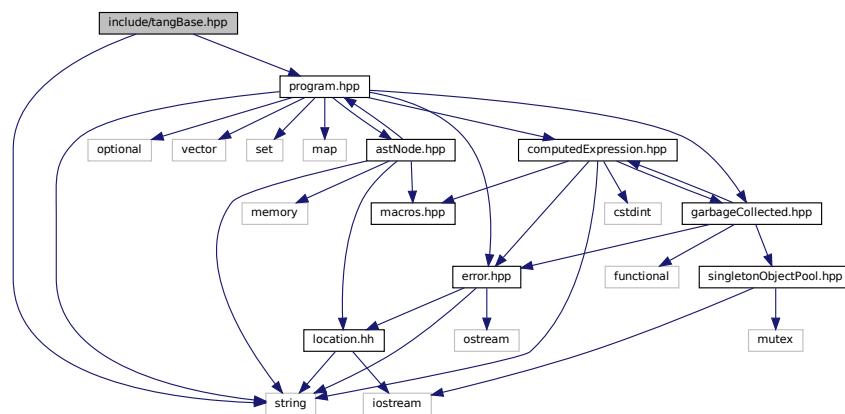
6.48.1 Detailed Description

Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

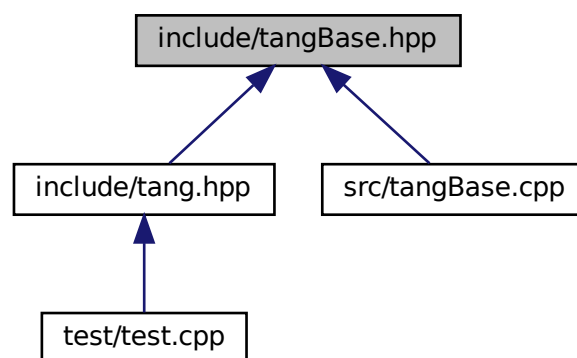
6.49 include/tangBase.hpp File Reference

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.

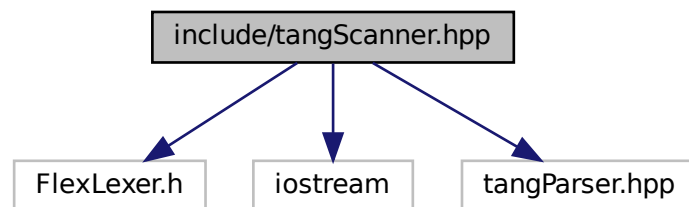
6.49.1 Detailed Description

Declare the [Tang::TangBase](#) class used to interact with Tang.

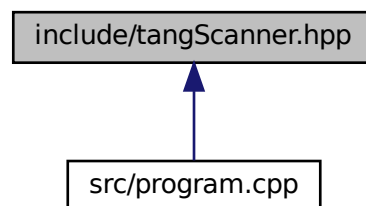
6.50 include/tangScanner.hpp File Reference

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\(\)](#)

6.50.1 Detailed Description

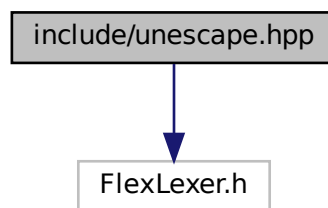
Declare the [Tang::TangScanner](#) used to tokenize a Tang script.

6.51 include/unescape.hpp File Reference

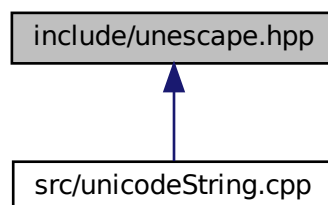
Declare the [Tang::Unescape](#) used to tokenize a Tang script.

```
#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\(\)](#)

6.51.1 Detailed Description

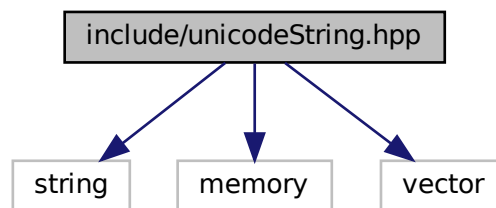
Declare the [Tang::Unescape](#) used to tokenize a Tang script.

6.52 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.

6.52.1 Detailed Description

Contains the code to interface with the ICU library.

6.52.2 Function Documentation

6.52.2.1 htmlEscape()

```
string Tang::htmlEscape (  
    const std::string & str )
```

Return an "html escaped" version of the provided string.

Only "critical" characters `<`, `>`, `&`, `"`, and ``` will be escaped. All other characters will be allowed through unaltered. The result is a UTF-8 encoded string that is safe for inclusion in an HTML template without disturbing the HTML structure.

Parameters

<i>str</i>	The string to be escaped.
------------	---------------------------

Returns

An "escaped" version of the provided string.

Here is the call graph for this function:



6.52.2.2 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:



6.52.2.3 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

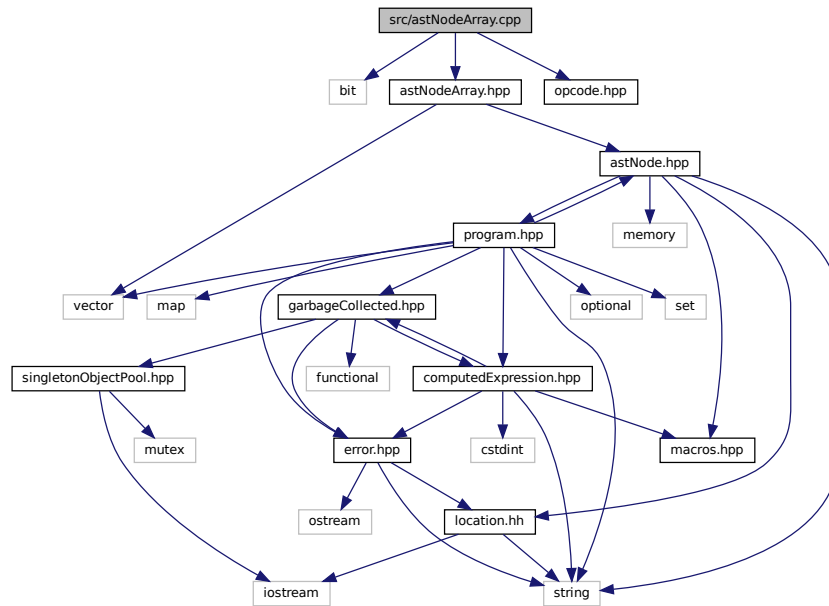
<i>str</i>	The string to be unescaped.
------------	-----------------------------

6.54 src/astNodeArray.cpp File Reference

Define the [Tang::AstNodeArray](#) class.

```
#include <bit>
#include "astNodeArray.hpp"
#include "opcode.hpp"
```

Include dependency graph for astNodeArray.cpp:



6.54.1 Detailed Description

Define the [Tang::AstNodeArray](#) class.

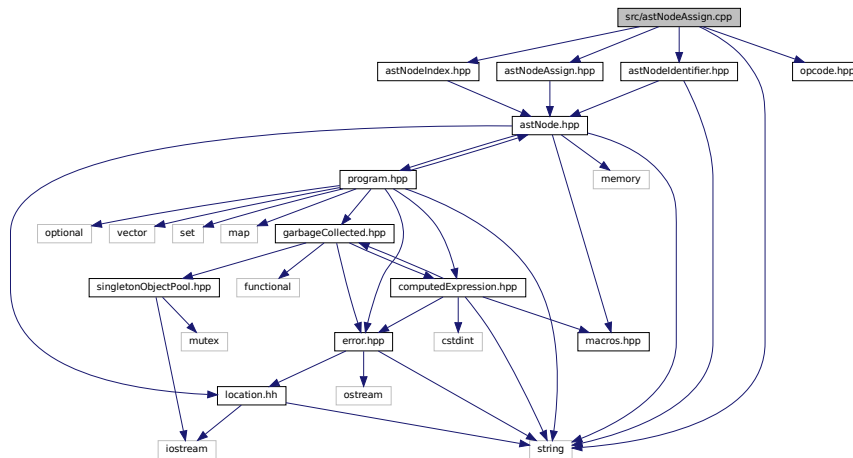
6.55 src/astNodeAssign.cpp File Reference

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:



6.55.1 Detailed Description

Define the [Tang::AstNodeAssign](#) class.

6.56 src/astNodeBinary.cpp File Reference

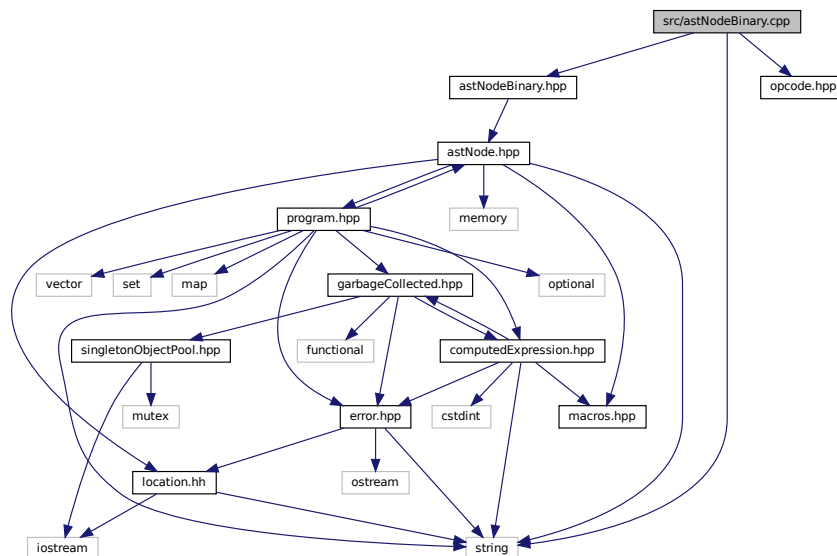
Define the [Tang::AstNodeBinary](#) class.

```
#include <string>
```

```
#include "astNodeBinary.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeBinary.cpp:



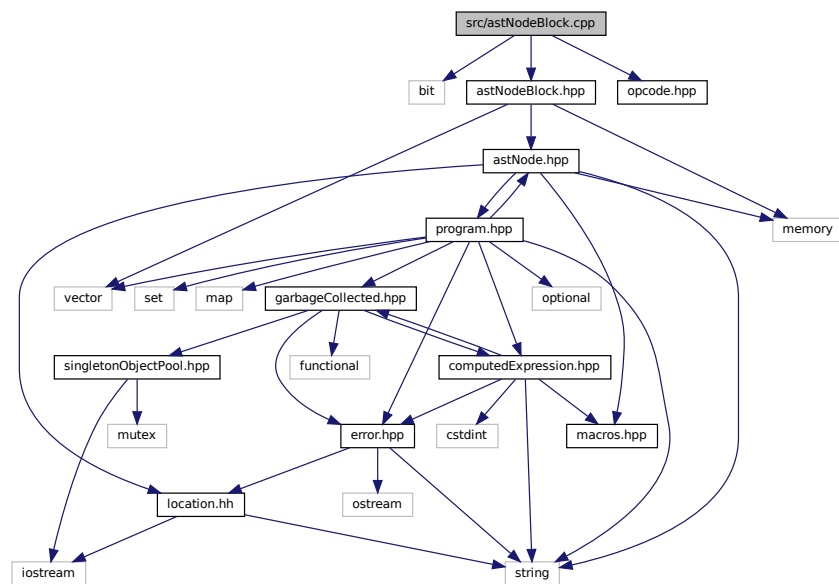
6.56.1 Detailed Description

Define the [Tang::AstNodeBinary](#) class.

6.57 src/astNodeBlock.cpp File Reference

Define the [Tang::AstNodeBlock](#) class.

```
#include <bit>
#include "astNodeBlock.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeBlock.cpp:
```



6.57.1 Detailed Description

Define the [Tang::AstNodeBlock](#) class.

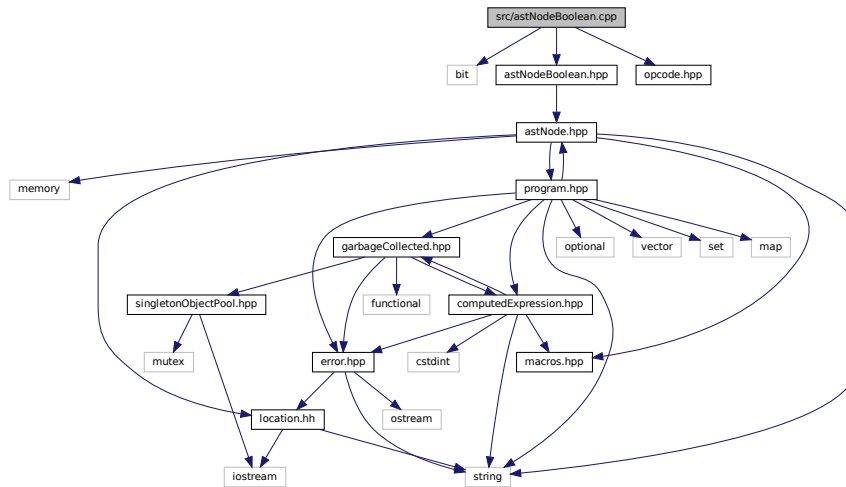
6.58 src/astNodeBoolean.cpp File Reference

Define the [Tang::AstNodeBoolean](#) class.

```
#include <bit>
#include "astNodeBoolean.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeBoolean.cpp:



6.58.1 Detailed Description

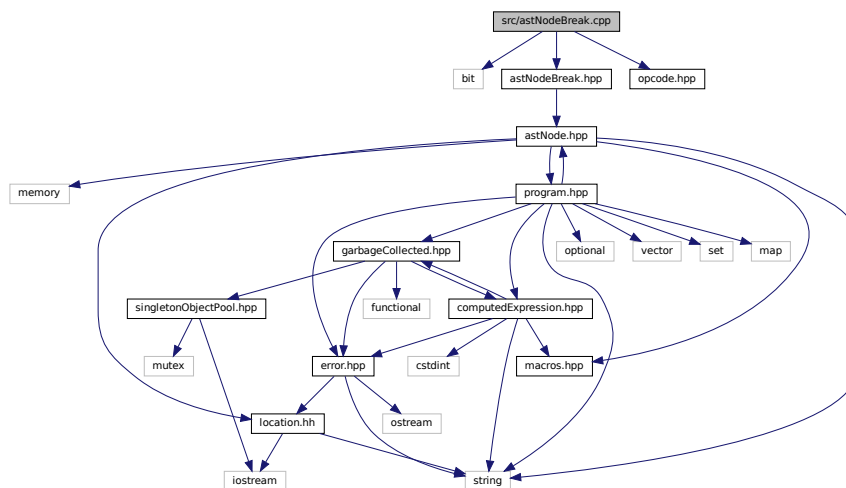
Define the [Tang::AstNodeBoolean](#) class.

6.59 src/astNodeBreak.cpp File Reference

Define the [Tang::AstNodeBreak](#) class.

```
#include <bit>
#include "astNodeBreak.hpp"
#include "opcode.hpp"
```

Include dependency graph for astNodeBreak.cpp:



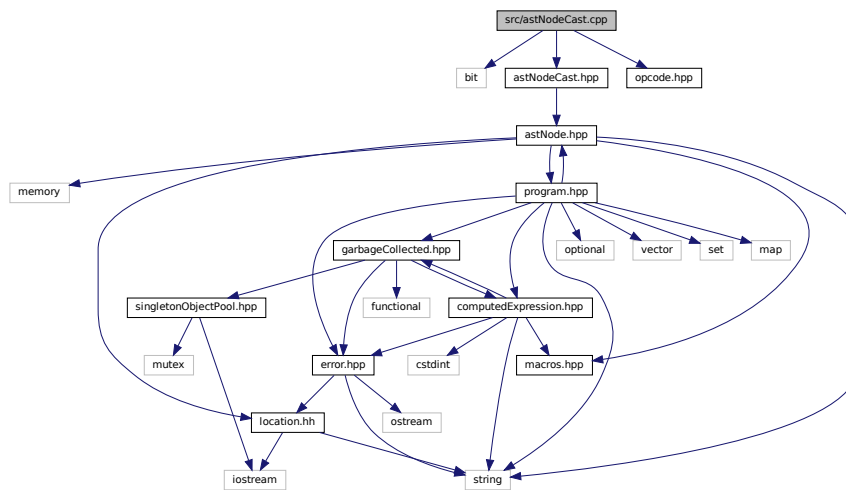
6.59.1 Detailed Description

Define the [Tang::AstNodeBreak](#) class.

6.60 src/astNodeCast.cpp File Reference

Define the [Tang::AstNodeCast](#) class.

```
#include <bit>
#include "astNodeCast.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeCast.cpp:
```



6.60.1 Detailed Description

Define the [Tang::AstNodeCast](#) class.

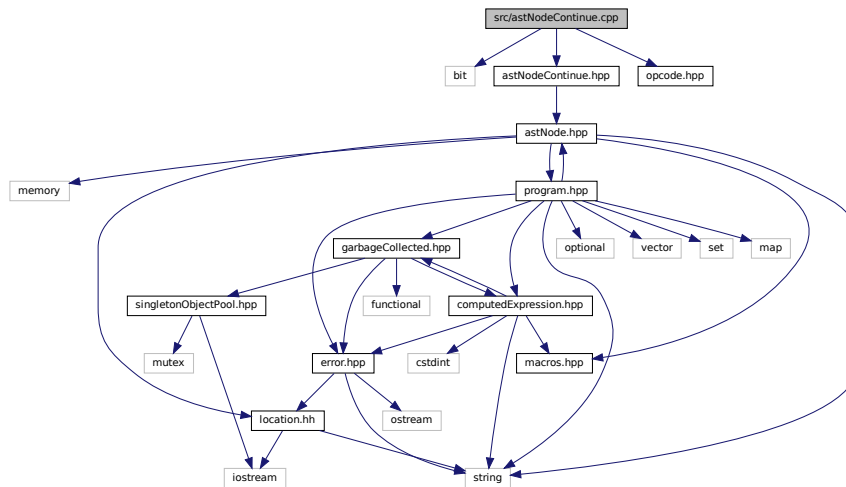
6.61 src/astNodeContinue.cpp File Reference

Define the [Tang::AstNodeContinue](#) class.

```
#include <bit>
#include "astNodeContinue.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for `astNodeContinue.cpp`:



6.61.1 Detailed Description

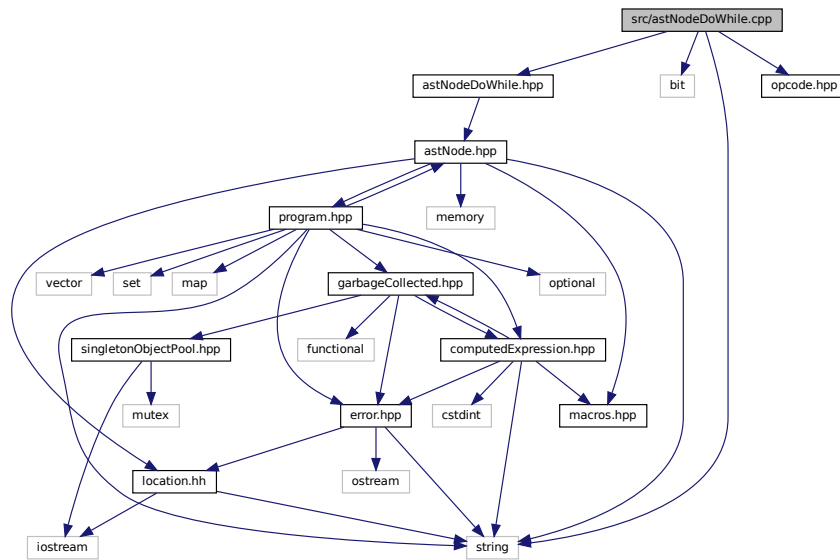
Define the [Tang::AstNodeContinue](#) class.

6.62 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:



6.62.1 Detailed Description

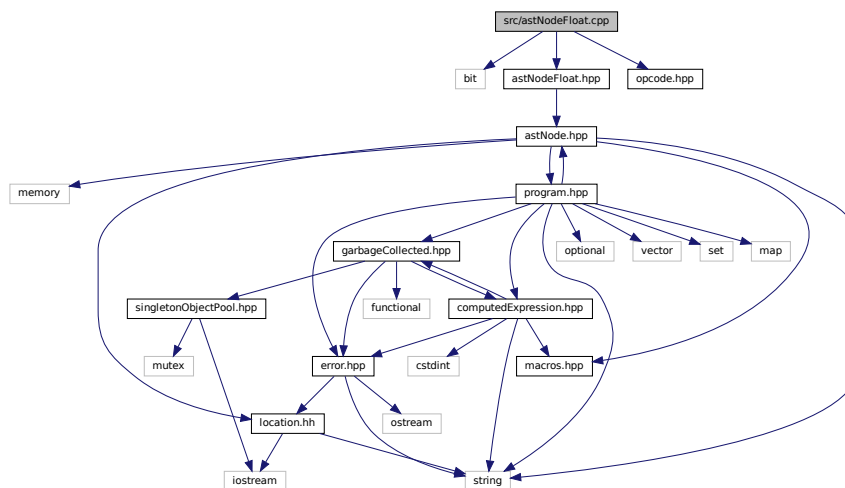
Define the [Tang::AstNodeDoWhile](#) class.

6.63 src/astNodeFloat.cpp File Reference

Define the [Tang::AstNodeFloat](#) class.

```
#include <bit>
#include "astNodeFloat.hpp"
#include "opcode.hpp"
```

Include dependency graph for astNodeFloat.cpp:



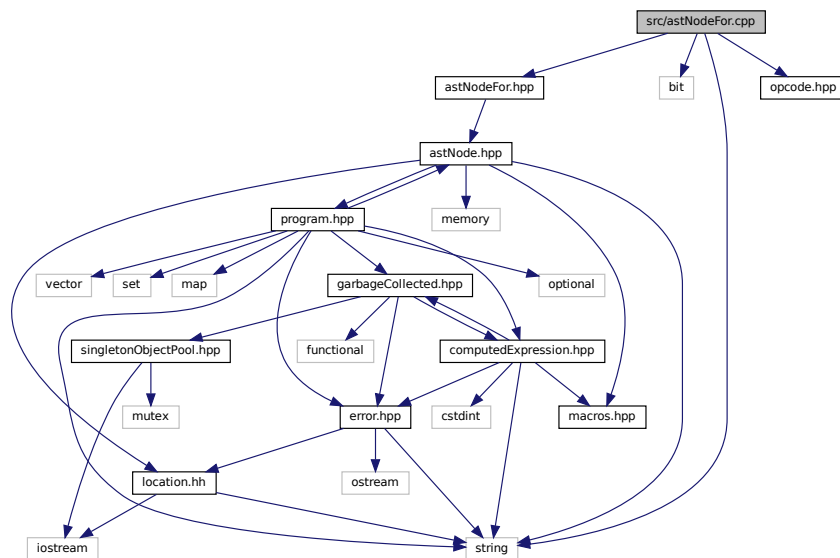
6.63.1 Detailed Description

Define the [Tang::AstNodeFloat](#) class.

6.64 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:
```



6.64.1 Detailed Description

Define the [Tang::AstNodeFor](#) class.

6.65 src/astNodeFunctionCall.cpp File Reference

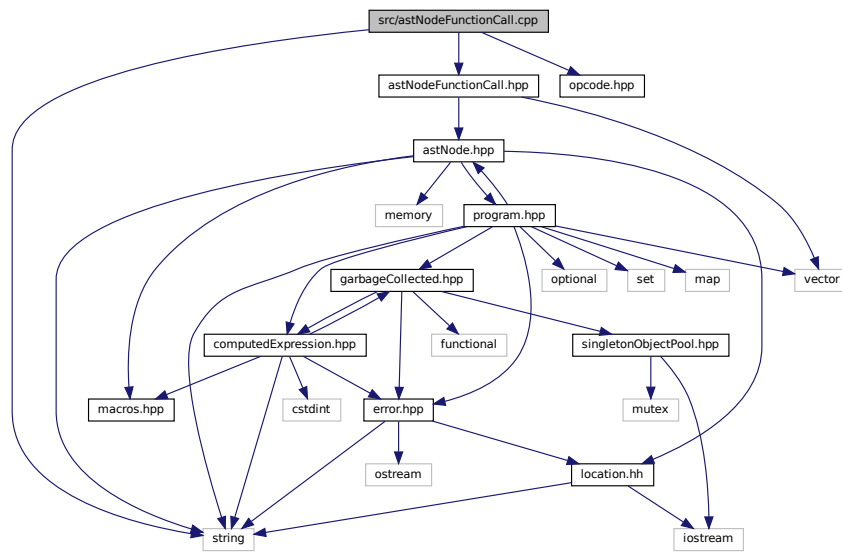
Define the [Tang::AstNodeFunctionCall](#) class.

```
#include <string>
#include "astNodeFunctionCall.hpp"
```



```
#include "opcode.hpp"
```

Include dependency graph for astNodeFunctionCall.cpp:



6.65.1 Detailed Description

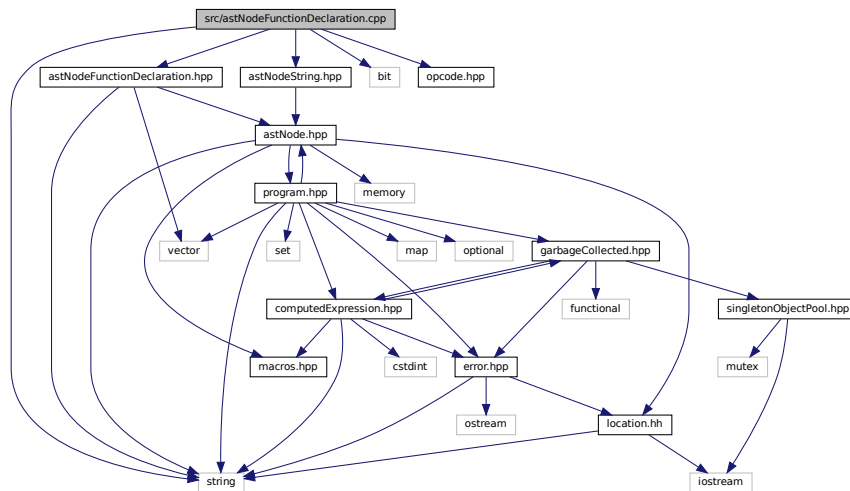
Define the [Tang::AstNodeFunctionCall](#) class.

6.66 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`:



6.66.1 Detailed Description

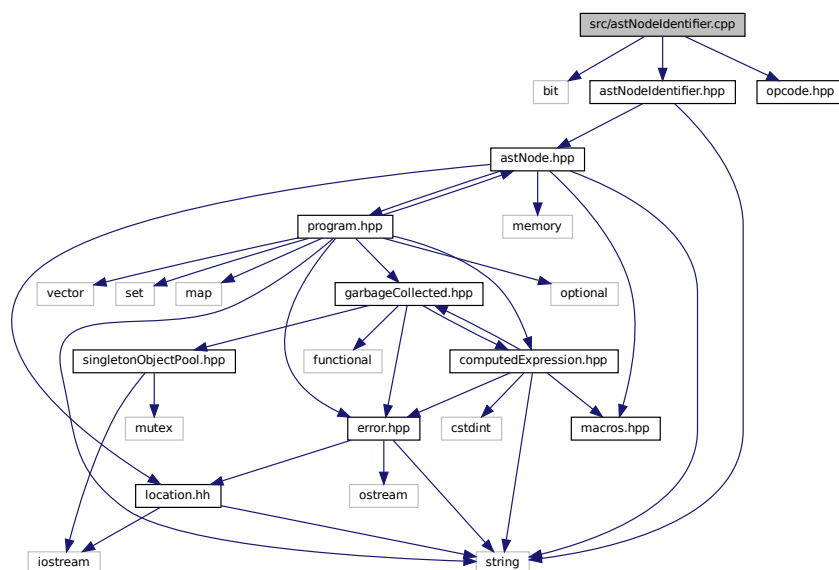
Define the [Tang::AstNodeFunctionDeclaration](#) class.

6.67 src/astNodeIdentifier.cpp File Reference

Define the [Tang::AstNodeIdentifier](#) class.

```
#include <bit>
#include "astNodeIdentifier.hpp"
#include "opcode.hpp"
```

Include dependency graph for `astNodeIdentifier.cpp`:



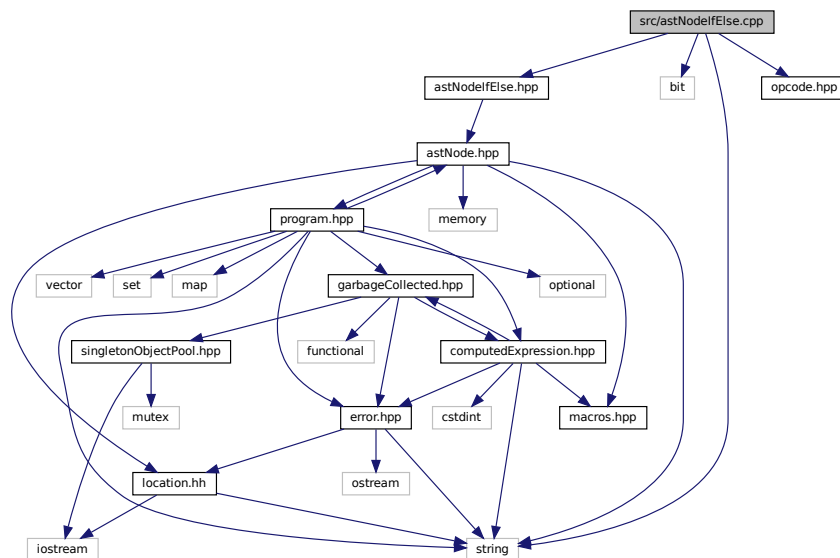
6.67.1 Detailed Description

Define the [Tang::AstNodeIdentifier](#) class.

6.68 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:
```



6.68.1 Detailed Description

Define the [Tang::AstNodeIfElse](#) class.

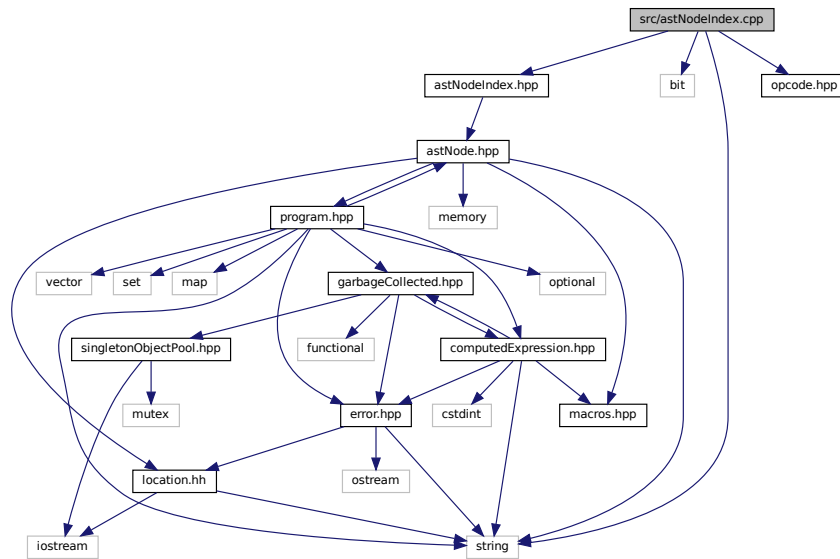
6.69 src/astNodeIndex.cpp File Reference

Define the [Tang::AstNodeIndex](#) class.

```
#include <string>
#include <bit>
#include "astNodeIndex.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeIndex.cpp:



6.69.1 Detailed Description

Define the [Tang::AstNodeIndex](#) class.

6.70 src/astNodeInteger.cpp File Reference

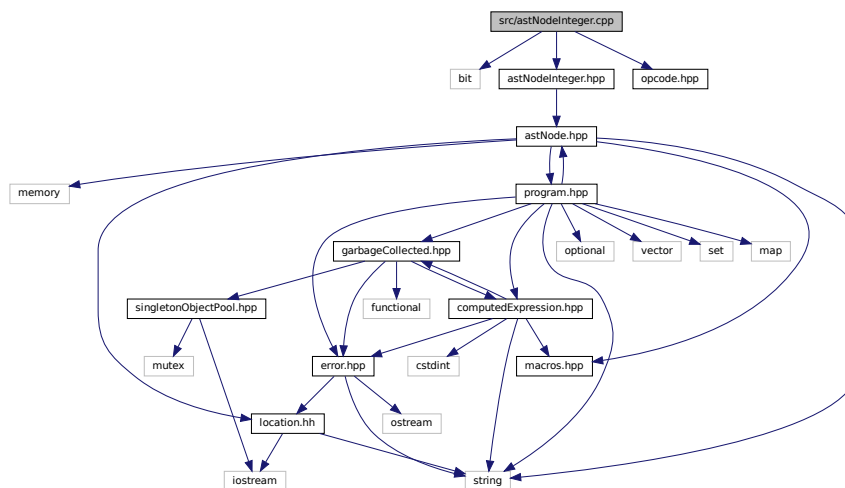
Define the [Tang::AstNodeInteger](#) class.

```
#include <bit>
```

```
#include "astNodeInteger.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodeInteger.cpp:



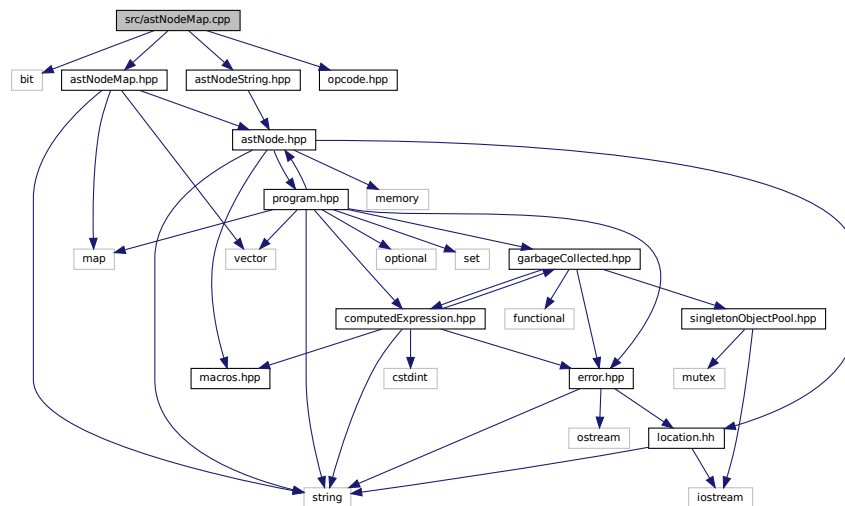
6.70.1 Detailed Description

Define the [Tang::AstNodeInteger](#) class.

6.71 src/astNodeMap.cpp File Reference

Define the [Tang::AstNodeMap](#) class.

```
#include <bit>
#include "astNodeMap.hpp"
#include "astNodeString.hpp"
#include "opcode.hpp"
Include dependency graph for astNodeMap.cpp:
```



6.71.1 Detailed Description

Define the [Tang::AstNodeMap](#) class.

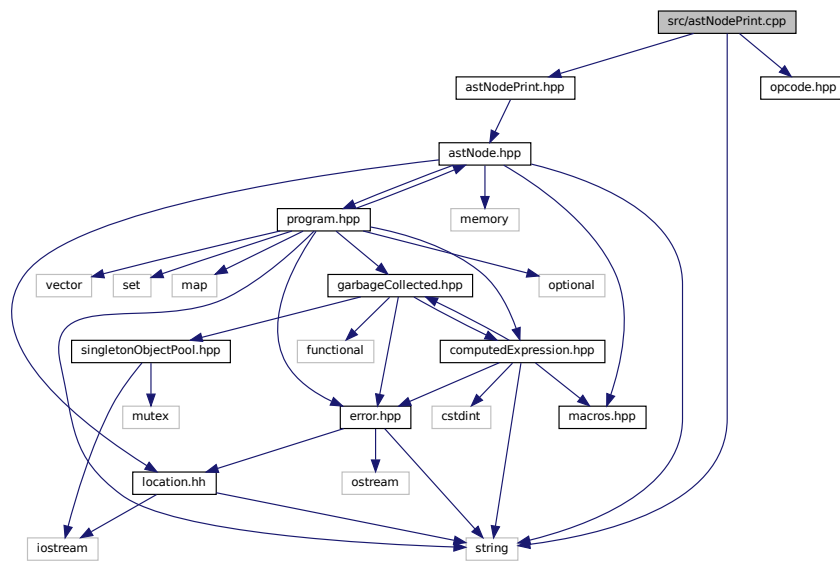
6.72 src/astNodePrint.cpp File Reference

Define the [Tang::AstNodePrint](#) class.

```
#include <string>
#include "astNodePrint.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for astNodePrint.cpp:



6.72.1 Detailed Description

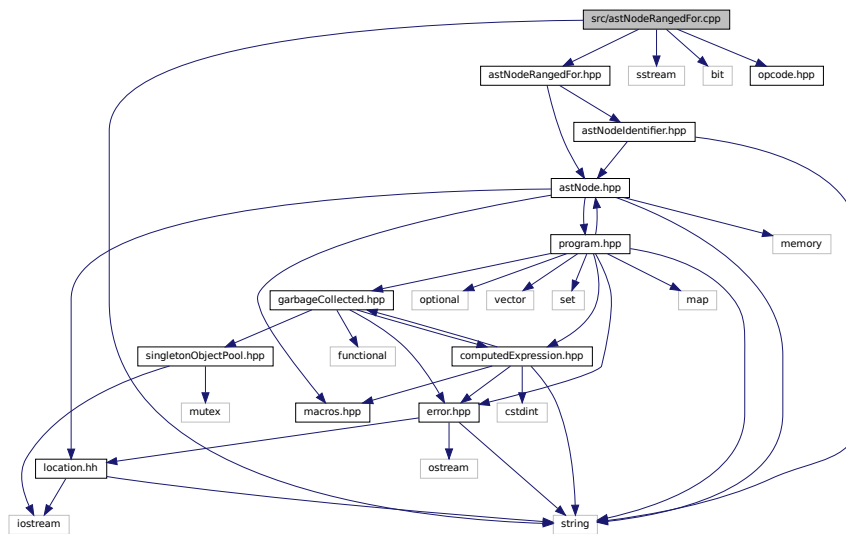
Define the [Tang::AstNodePrint](#) class.

6.73 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:



6.73.1 Detailed Description

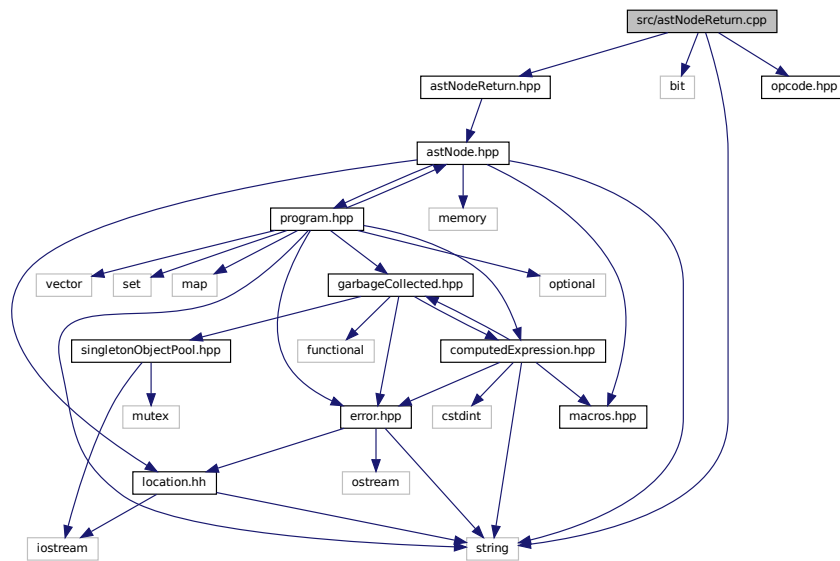
Define the [Tang::AstNodeRangedFor](#) class.

6.74 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`:



6.74.1 Detailed Description

Define the [Tang::AstNodeReturn](#) class.

6.75 src/astNodeSlice.cpp File Reference

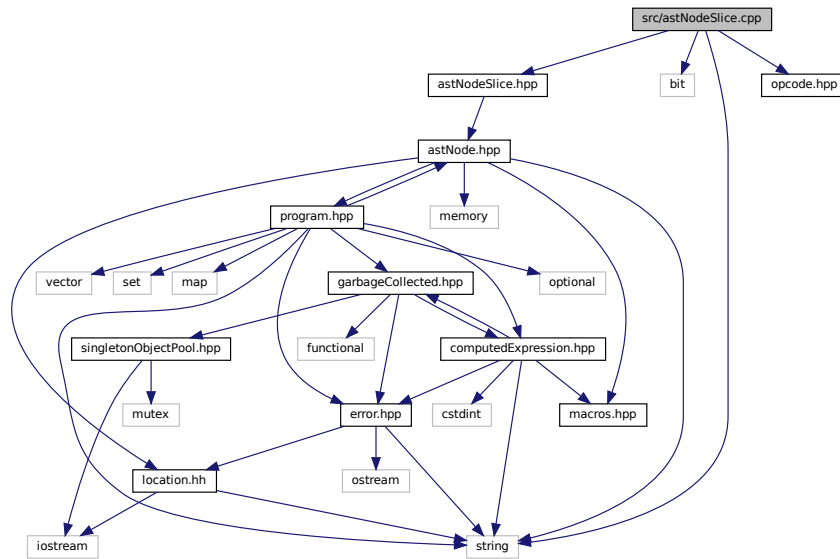
Define the [Tang::AstNodeSlice](#) class.

```

#include <string>
#include <bit>
#include "astNodeSlice.hpp"
#include "opcode.hpp"

```


Include dependency graph for astNodeSlice.cpp:



6.75.1 Detailed Description

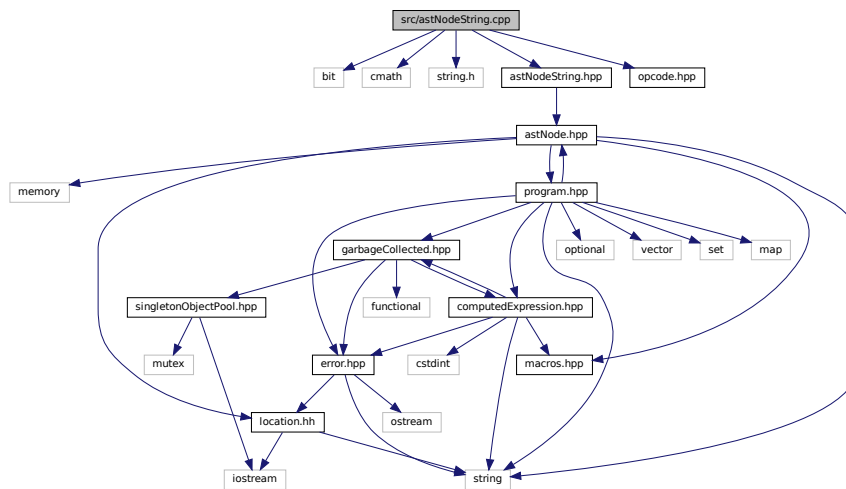
Define the [Tang::AstNodeSlice](#) class.

6.76 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:



6.76.1 Detailed Description

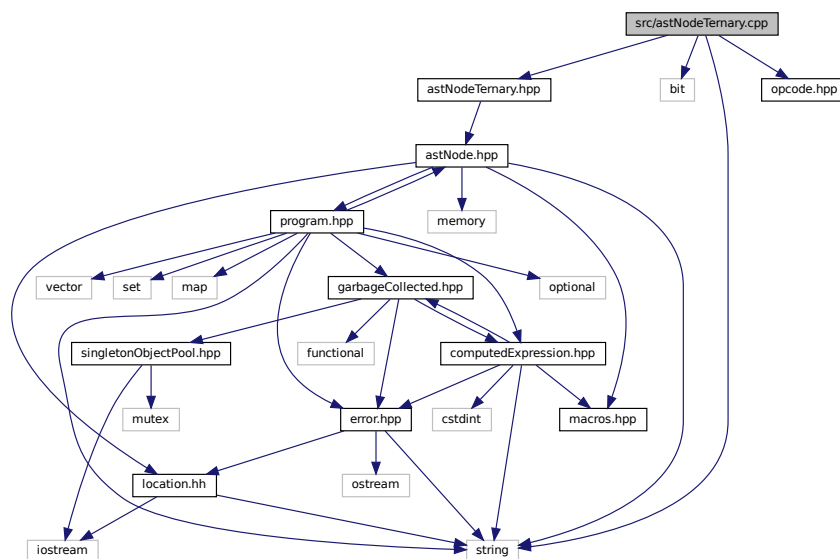
Define the [Tang::AstNodeString](#) class.

6.77 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:



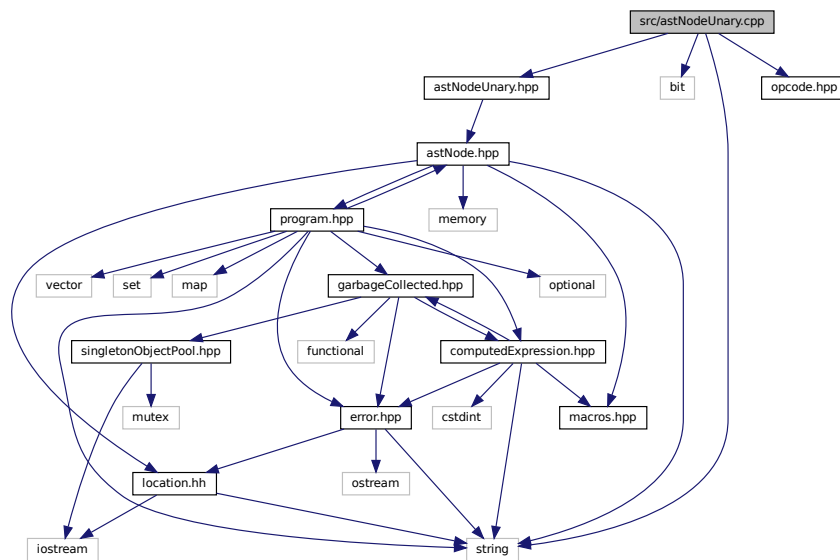
6.77.1 Detailed Description

Define the [Tang::AstNodeUnary](#) class.

6.78 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:
```



6.78.1 Detailed Description

Define the [Tang::AstNodeUnary](#) class.

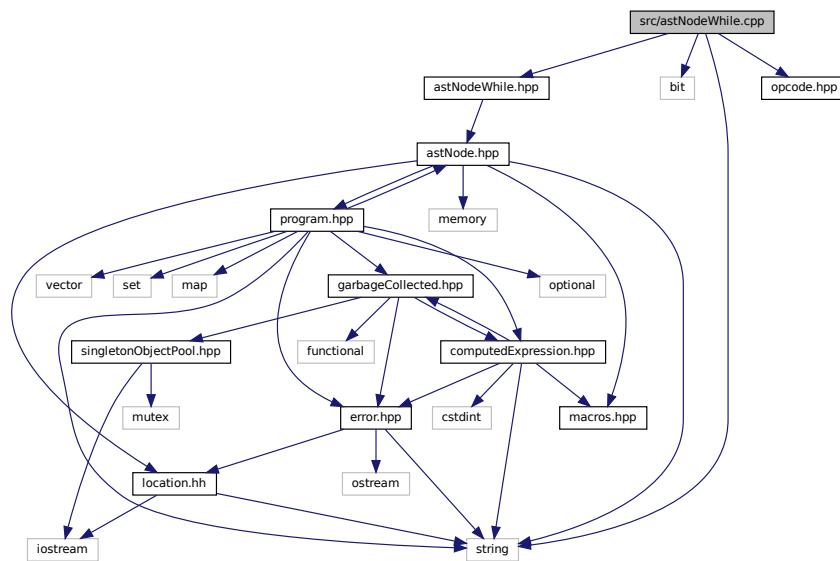
6.79 src/astNodeWhile.cpp File Reference

Define the [Tang::AstNodeWhile](#) class.

```
#include <string>
#include <bit>
#include "astNodeWhile.hpp"
```

```
#include "opcode.hpp"
```

Include dependency graph for `astNodeWhile.cpp`:



6.79.1 Detailed Description

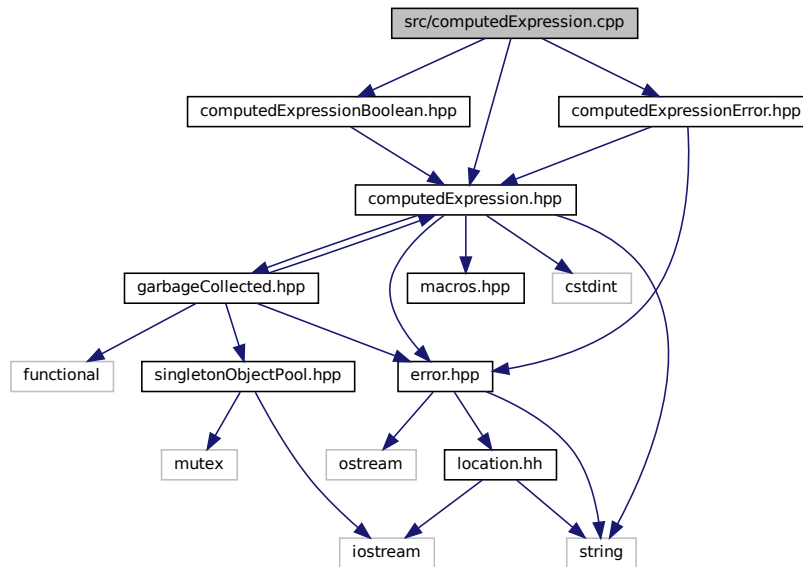
Define the [Tang::AstNodeWhile](#) class.

6.80 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:



6.80.1 Detailed Description

Define the [Tang::ComputedExpression](#) class.

6.81 src/computedExpressionArray.cpp File Reference

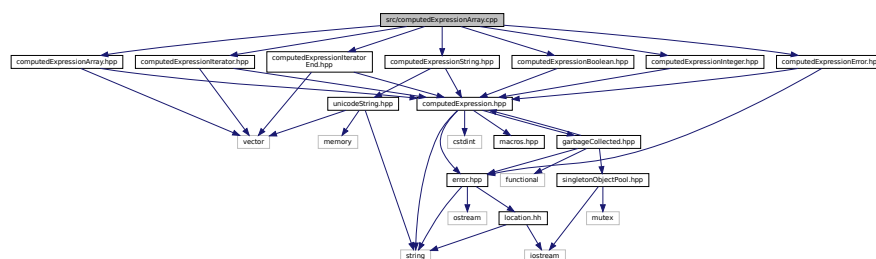
Define the [Tang::ComputedExpressionArray](#) class.

```

#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:



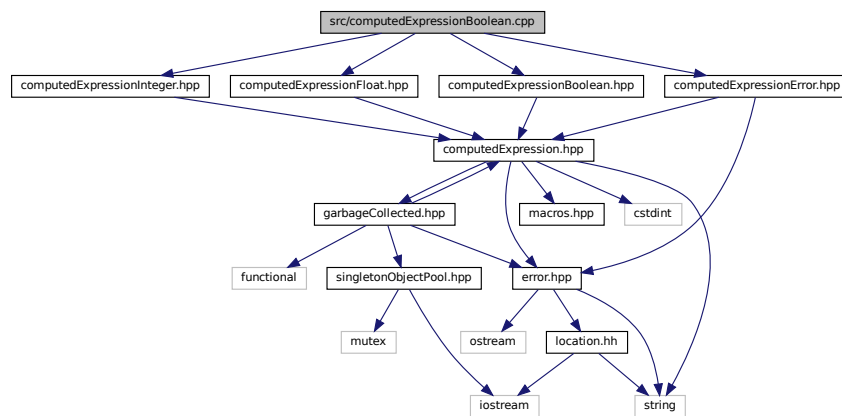
6.81.1 Detailed Description

Define the [Tang::ComputedExpressionArray](#) class.

6.82 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:
```



6.82.1 Detailed Description

Define the [Tang::ComputedExpressionBoolean](#) class.

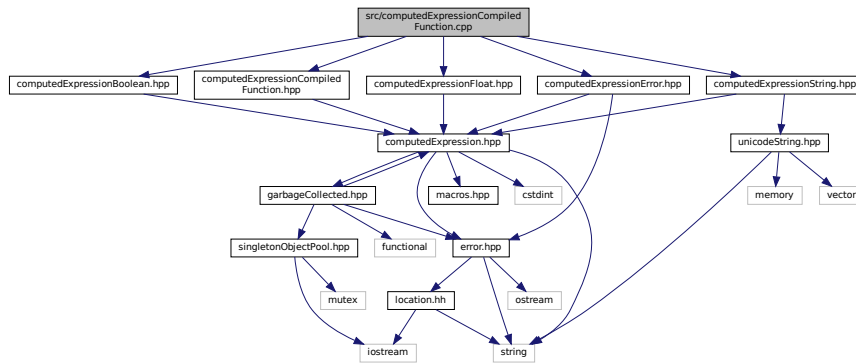
6.83 src/computedExpressionCompiledFunction.cpp File Reference

Define the [Tang::ComputedExpressionCompiledFunction](#) class.

```
#include "computedExpressionCompiledFunction.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
```

```
#include "computedExpressionError.hpp"
```

Include dependency graph for computedExpressionCompiledFunction.cpp:



6.83.1 Detailed Description

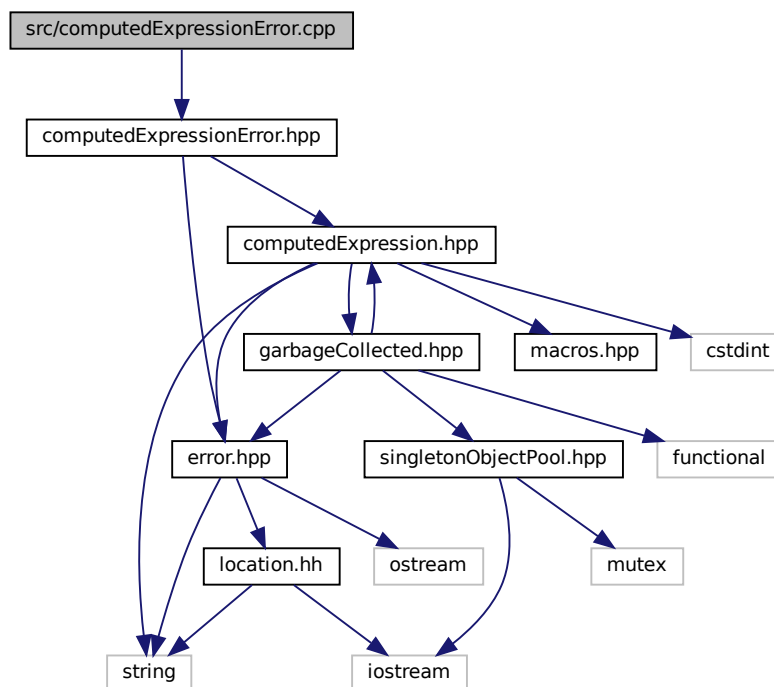
Define the [Tang::ComputedExpressionCompiledFunction](#) class.

6.84 src/computedExpressionError.cpp File Reference

Define the [Tang::ComputedExpressionError](#) class.

```
#include "computedExpressionError.hpp"
```

Include dependency graph for computedExpressionError.cpp:



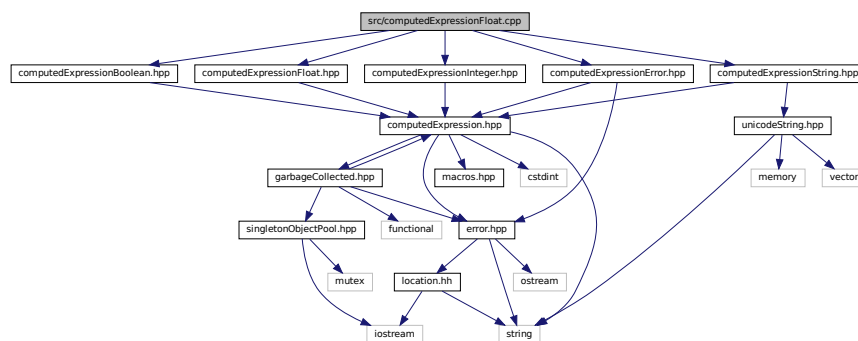
6.84.1 Detailed Description

Define the [Tang::ComputedExpressionError](#) class.

6.85 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:
```



6.85.1 Detailed Description

Define the [Tang::ComputedExpressionFloat](#) class.

6.86 src/computedExpressionInteger.cpp File Reference

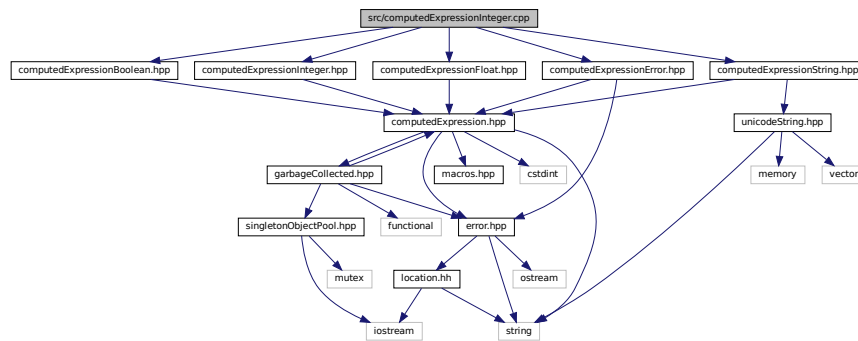
Define the [Tang::ComputedExpressionInteger](#) class.

```
#include "computedExpressionInteger.hpp"
#include "computedExpressionFloat.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
```



```
#include "computedExpressionError.hpp"
```

Include dependency graph for `computedExpressionInteger.cpp`:



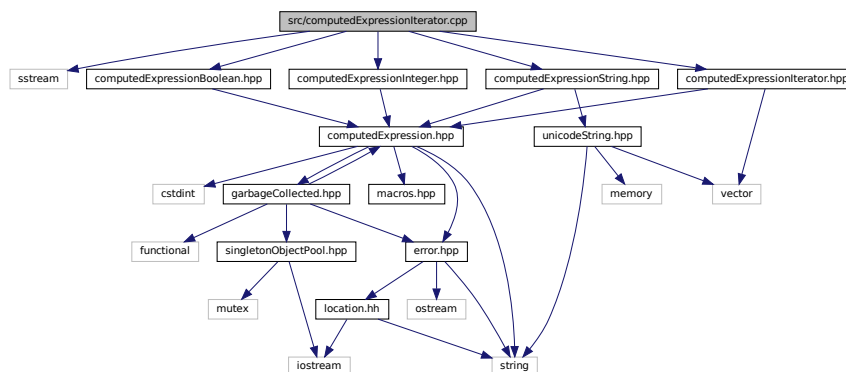
6.86.1 Detailed Description

Define the `Tang::ComputedExpressionInteger` class.

6.87 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:
```



6.87.1 Detailed Description

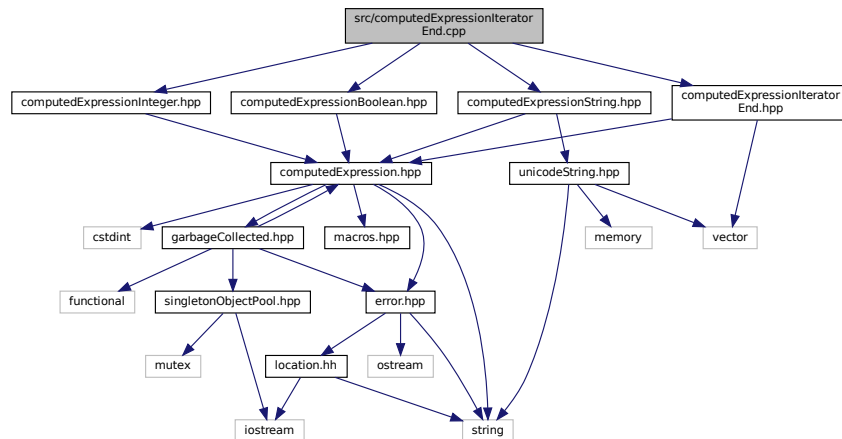
Define the `Tang::ComputedExpressionIterator` class.

6.88 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:



6.88.1 Detailed Description

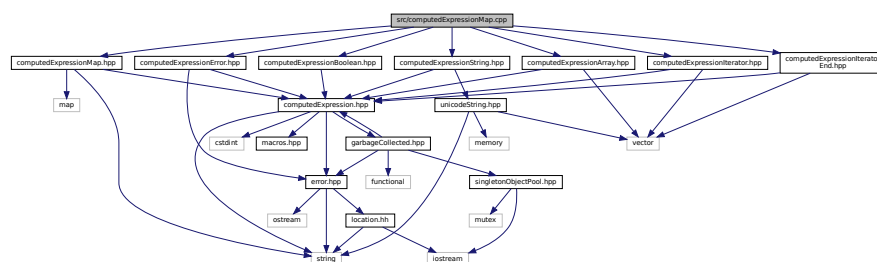
Define the [Tang::ComputedExpressionIteratorEnd](#) class.

6.89 src/computedExpressionMap.cpp File Reference

Define the [Tang::ComputedExpressionMap](#) class.

```
#include "computedExpressionMap.hpp"
#include "computedExpressionArray.hpp"
#include "computedExpressionBoolean.hpp"
#include "computedExpressionString.hpp"
#include "computedExpressionIterator.hpp"
#include "computedExpressionIteratorEnd.hpp"
#include "computedExpressionError.hpp"
```

Include dependency graph for computedExpressionMap.cpp:



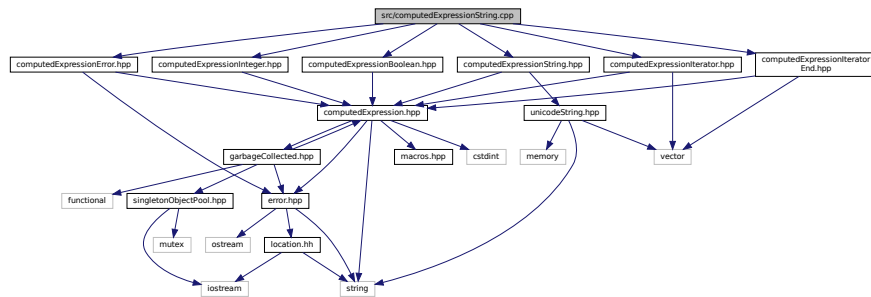
6.89.1 Detailed Description

Define the [Tang::ComputedExpressionMap](#) class.

6.90 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:
```



6.90.1 Detailed Description

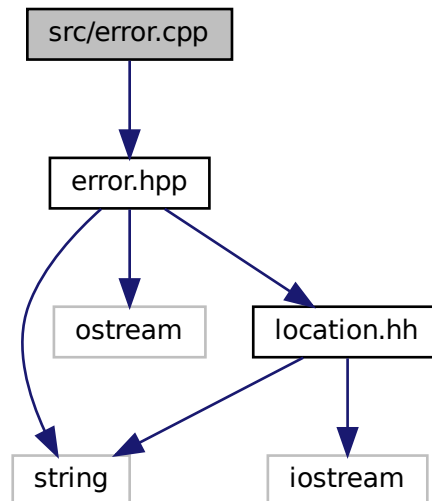
Define the [Tang::ComputedExpressionString](#) class.

6.91 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)`

6.91.1 Detailed Description

Define the `Tang::Error` class.

6.91.2 Function Documentation

6.91.2.1 `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

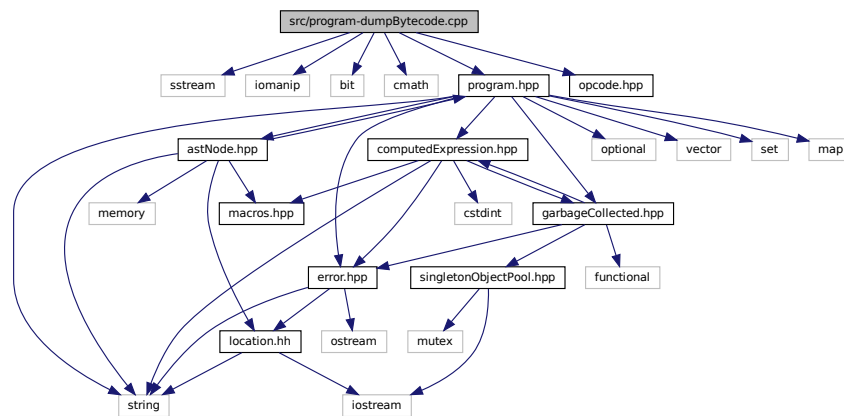
The output stream.

6.92 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.

6.92.1 Detailed Description

Define the [Tang::Program::dumpBytecode](#) method.

6.92.2 Macro Definition Documentation

6.92.2.1 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

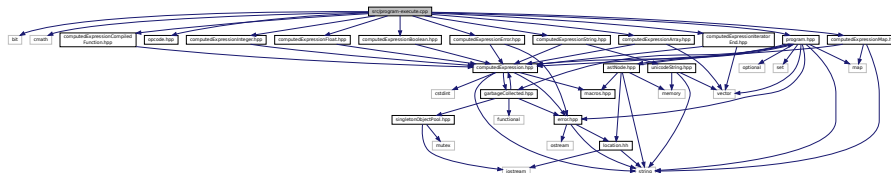
x	The number of additional vector entries that should exist.
---	--

6.93 src/program-execute.cpp File Reference

Define the [Tang::Program::execute](#) method.

```
#include <bit>  
#include <cmath>  
#include "program.hpp"  
#include "opcode.hpp"  
#include "computedExpressionError.hpp"  
#include "computedExpressionInteger.hpp"  
#include "computedExpressionFloat.hpp"  
#include "computedExpressionBoolean.hpp"  
#include "computedExpressionString.hpp"  
#include "computedExpressionArray.hpp"  
#include "computedExpressionMap.hpp"  
#include "computedExpressionCompiledFunction.hpp"  
#include "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.

6.93.1 Detailed Description

Define the [Tang::Program::execute](#) method.

6.93.2 Macro Definition Documentation

6.93.2.1 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.
---	--

6.93.2.2 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.
---	---

6.94 src/program.cpp File Reference

Define the [Tang::Program](#) class.

6.95.1 Detailed Description

Define the [Tang::TangBase](#) class.

6.96 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:
```



6.96.1 Detailed Description

Contains the function declarations for the [Tang::UnicodeString](#) class and the interface to ICU.

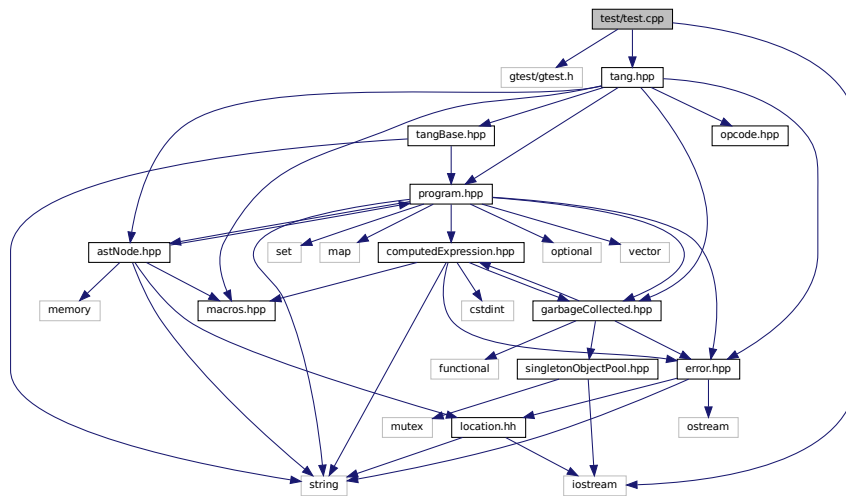
6.97 test/test.cpp File Reference

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** (Expression, Map)
- **TEST** (CodeBlock, Statements)
- **TEST** (Assign, Identifier)
- **TEST** (Assign, Index)

- **TEST** (Expression, ArraySlice)
- **TEST** (ControlFlow, IfElse)
- **TEST** (ControlFlow, While)
- **TEST** (ControlFlow, Break)
- **TEST** (ControlFlow, Continue)
- **TEST** (ControlFlow, DoWhile)
- **TEST** (ControlFlow, For)
- **TEST** (ControlFlow, RangedFor)
- **TEST** (Print, Default)
- **TEST** (Print, Array)

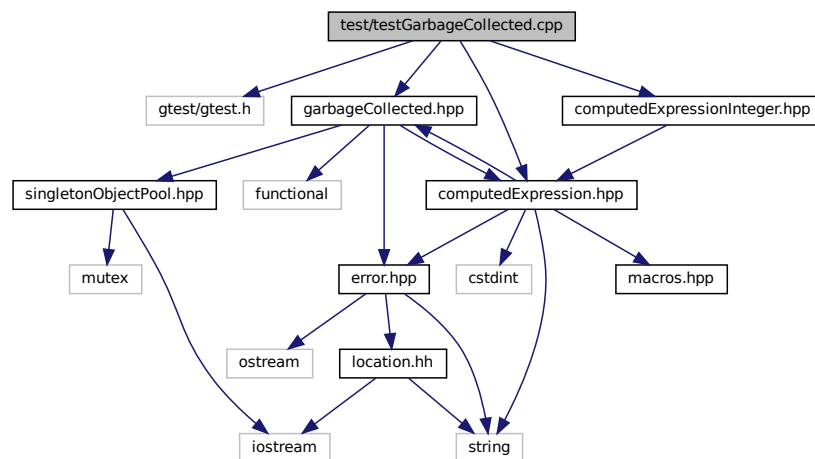
6.97.1 Detailed Description

Test the general language behaviors.

6.98 test/testGarbageCollected.cpp File Reference

Test the generic behavior of the [Tang::GarbageCollected](#) class.

```
#include <gtest/gtest.h>
#include "garbageCollected.hpp"
#include "computedExpression.hpp"
#include "computedExpressionInteger.hpp"
Include dependency graph for testGarbageCollected.cpp:
```



Functions

- **TEST** (Create, Access)
- **TEST** (RuleOfFive, CopyConstructor)
- **TEST** (Recycle, ObjectIsRecycled)
- **TEST** (Recycle, ObjectIsNotRecycled)
- `int main` (int argc, char **argv)

6.98.1 Detailed Description

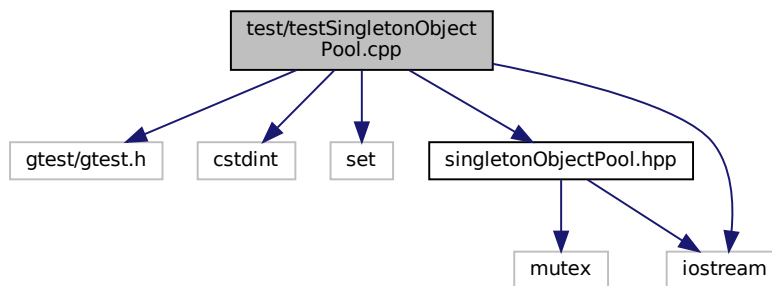
Test the generic behavior of the [Tang::GarbageCollected](#) class.

6.99 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`)

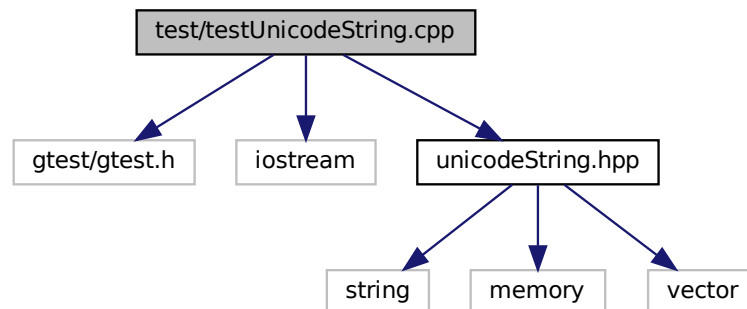
6.99.1 Detailed Description

Test the generic behavior of the [Tang::SingletonObjectPool](#) class.

6.100 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)

6.100.1 Detailed Description

Contains tests for the [Tang::UnicodeString](#) class.

Index

- __add
 - Tang::ComputedExpression, 137
 - Tang::ComputedExpressionArray, 150
 - Tang::ComputedExpressionBoolean, 163
 - Tang::ComputedExpressionCompiledFunction, 175
 - Tang::ComputedExpressionError, 189
 - Tang::ComputedExpressionFloat, 201
 - Tang::ComputedExpressionInteger, 216
 - Tang::ComputedExpressionIterator, 232
 - Tang::ComputedExpressionIteratorEnd, 245
 - Tang::ComputedExpressionMap, 258
 - Tang::ComputedExpressionString, 272
- __asCode
 - Tang::ComputedExpression, 137
 - Tang::ComputedExpressionArray, 150
 - Tang::ComputedExpressionBoolean, 163
 - Tang::ComputedExpressionCompiledFunction, 176
 - Tang::ComputedExpressionError, 189
 - Tang::ComputedExpressionFloat, 203
 - Tang::ComputedExpressionInteger, 218
 - Tang::ComputedExpressionIterator, 232
 - Tang::ComputedExpressionIteratorEnd, 245
 - Tang::ComputedExpressionMap, 258
 - Tang::ComputedExpressionString, 272
- __assign_index
 - Tang::ComputedExpression, 137
 - Tang::ComputedExpressionArray, 150
 - Tang::ComputedExpressionBoolean, 164
 - Tang::ComputedExpressionCompiledFunction, 176
 - Tang::ComputedExpressionError, 190
 - Tang::ComputedExpressionFloat, 203
 - Tang::ComputedExpressionInteger, 218
 - Tang::ComputedExpressionIterator, 232
 - Tang::ComputedExpressionIteratorEnd, 245
 - Tang::ComputedExpressionMap, 258
 - Tang::ComputedExpressionString, 273
- __boolean
 - Tang::ComputedExpression, 138
 - Tang::ComputedExpressionArray, 151
 - Tang::ComputedExpressionBoolean, 164
 - Tang::ComputedExpressionCompiledFunction, 176
 - Tang::ComputedExpressionError, 190
 - Tang::ComputedExpressionFloat, 204
 - Tang::ComputedExpressionInteger, 219
 - Tang::ComputedExpressionIterator, 233
 - Tang::ComputedExpressionIteratorEnd, 246
 - Tang::ComputedExpressionMap, 259
 - Tang::ComputedExpressionString, 273
- __divide
 - Tang::ComputedExpression, 138
 - Tang::ComputedExpressionArray, 151
 - Tang::ComputedExpressionBoolean, 164
 - Tang::ComputedExpressionCompiledFunction, 177
 - Tang::ComputedExpressionError, 190
 - Tang::ComputedExpressionFloat, 204
 - Tang::ComputedExpressionInteger, 219
 - Tang::ComputedExpressionIterator, 233
 - Tang::ComputedExpressionIteratorEnd, 246
 - Tang::ComputedExpressionMap, 259
 - Tang::ComputedExpressionString, 274
- __equal
 - Tang::ComputedExpression, 138
 - Tang::ComputedExpressionArray, 152
 - Tang::ComputedExpressionBoolean, 165
 - Tang::ComputedExpressionCompiledFunction, 177
 - Tang::ComputedExpressionError, 191
 - Tang::ComputedExpressionFloat, 204
 - Tang::ComputedExpressionInteger, 219
 - Tang::ComputedExpressionIterator, 234
 - Tang::ComputedExpressionIteratorEnd, 247
 - Tang::ComputedExpressionMap, 260
 - Tang::ComputedExpressionString, 274
- __float
 - Tang::ComputedExpression, 139
 - Tang::ComputedExpressionArray, 152
 - Tang::ComputedExpressionBoolean, 165
 - Tang::ComputedExpressionCompiledFunction, 178
 - Tang::ComputedExpressionError, 191
 - Tang::ComputedExpressionFloat, 205
 - Tang::ComputedExpressionInteger, 220
 - Tang::ComputedExpressionIterator, 234
 - Tang::ComputedExpressionIteratorEnd, 247
 - Tang::ComputedExpressionMap, 260
 - Tang::ComputedExpressionString, 275
- __getIterator
 - Tang::ComputedExpression, 139
 - Tang::ComputedExpressionArray, 152
 - Tang::ComputedExpressionBoolean, 165
 - Tang::ComputedExpressionCompiledFunction, 178
 - Tang::ComputedExpressionError, 191
 - Tang::ComputedExpressionFloat, 205
 - Tang::ComputedExpressionInteger, 220
 - Tang::ComputedExpressionIterator, 234
 - Tang::ComputedExpressionIteratorEnd, 247
 - Tang::ComputedExpressionMap, 260
 - Tang::ComputedExpressionString, 275
- __index
 - Tang::ComputedExpression, 139

- Tang::ComputedExpressionMap, 264
- Tang::ComputedExpressionString, 280
- __subtract
 - Tang::ComputedExpression, 143
 - Tang::ComputedExpressionArray, 157
 - Tang::ComputedExpressionBoolean, 169
 - Tang::ComputedExpressionCompiledFunction, 182
 - Tang::ComputedExpressionError, 195
 - Tang::ComputedExpressionFloat, 210
 - Tang::ComputedExpressionInteger, 225
 - Tang::ComputedExpressionIterator, 238
 - Tang::ComputedExpressionIteratorEnd, 251
 - Tang::ComputedExpressionMap, 264
 - Tang::ComputedExpressionString, 280
- ~GarbageCollected
 - Tang::GarbageCollected, 289
- ADD
 - opcode.hpp, 380
- Add
 - Tang::AstNodeBinary, 30
- addBreak
 - Tang::Program, 314
- addBytecode
 - Tang::Program, 314
- addContinue
 - Tang::Program, 314
- addIdentifier
 - Tang::Program, 315
- addIdentifierAssigned
 - Tang::Program, 315
- addString
 - Tang::Program, 315
- And
 - Tang::AstNodeBinary, 30
- ARRAY
 - opcode.hpp, 380
- ASSIGNINDEX
 - opcode.hpp, 380
- AstNode
 - Tang::AstNode, 16
- AstNodeArray
 - Tang::AstNodeArray, 21
- AstNodeAssign
 - Tang::AstNodeAssign, 25
- AstNodeBinary
 - Tang::AstNodeBinary, 31
- AstNodeBlock
 - Tang::AstNodeBlock, 35
- AstNodeBoolean
 - Tang::AstNodeBoolean, 39
- AstNodeBreak
 - Tang::AstNodeBreak, 43
- AstNodeCast
 - Tang::AstNodeCast, 48
- AstNodeContinue
 - Tang::AstNodeContinue, 52
- AstNodeDoWhile
 - Tang::AstNodeDoWhile, 56
- AstNodeFloat
 - Tang::AstNodeFloat, 60
- AstNodeFor
 - Tang::AstNodeFor, 65
- AstNodeFunctionCall
 - Tang::AstNodeFunctionCall, 69
- AstNodeFunctionDeclaration
 - Tang::AstNodeFunctionDeclaration, 72
- AstNodeIdentifier
 - Tang::AstNodeIdentifier, 77
- AstNodeIfElse
 - Tang::AstNodeIfElse, 82
- AstNodeIndex
 - Tang::AstNodeIndex, 86
- AstNodeInteger
 - Tang::AstNodeInteger, 91
- AstNodeMap
 - Tang::AstNodeMap, 95
- AstNodePrint
 - Tang::AstNodePrint, 100
- AstNodeRangedFor
 - Tang::AstNodeRangedFor, 103
- AstNodeReturn
 - Tang::AstNodeReturn, 108
- AstNodeSlice
 - Tang::AstNodeSlice, 113
- AstNodeString
 - Tang::AstNodeString, 117
- AstNodeTernary
 - Tang::AstNodeTernary, 122
- AstNodeUnary
 - Tang::AstNodeUnary, 127
- AstNodeWhile
 - Tang::AstNodeWhile, 132
- BOOLEAN
 - opcode.hpp, 380
- Boolean
 - Tang::AstNodeCast, 48
- build/generated/location.hh, 335
- bytesLength
 - Tang::UnicodeString, 331
- CALLFUNC
 - opcode.hpp, 381
- CASTBOOLEAN
 - opcode.hpp, 381
- CASTFLOAT
 - opcode.hpp, 381
- CASTINTEGER
 - opcode.hpp, 381
- CASTSTRING
 - opcode.hpp, 381
- CodeType
 - Tang::Program, 313
- compile
 - Tang::AstNode, 17
 - Tang::AstNodeArray, 22
 - Tang::AstNodeAssign, 26

- Tang::AstNodeBinary, 31
- Tang::AstNodeBlock, 36
- Tang::AstNodeBoolean, 40
- Tang::AstNodeBreak, 43
- Tang::AstNodeCast, 48
- Tang::AstNodeContinue, 52
- Tang::AstNodeDoWhile, 57
- Tang::AstNodeFloat, 61
- Tang::AstNodeFor, 65
- Tang::AstNodeFunctionCall, 69
- Tang::AstNodeFunctionDeclaration, 73
- Tang::AstNodeIdentifier, 77
- Tang::AstNodeIfElse, 82
- Tang::AstNodeIndex, 87
- Tang::AstNodeInteger, 92
- Tang::AstNodeMap, 95
- Tang::AstNodePrint, 100
- Tang::AstNodeRangedFor, 104
- Tang::AstNodeReturn, 109
- Tang::AstNodeSlice, 113
- Tang::AstNodeString, 118
- Tang::AstNodeTernary, 123
- Tang::AstNodeUnary, 127
- Tang::AstNodeWhile, 133
- compileLiteral
 - Tang::AstNodeString, 118
- compilePreprocess
 - Tang::AstNode, 17
 - Tang::AstNodeArray, 22
 - Tang::AstNodeAssign, 26
 - Tang::AstNodeBinary, 32
 - Tang::AstNodeBlock, 36
 - Tang::AstNodeBoolean, 40
 - Tang::AstNodeBreak, 44
 - Tang::AstNodeCast, 49
 - Tang::AstNodeContinue, 53
 - Tang::AstNodeDoWhile, 57
 - Tang::AstNodeFloat, 61
 - Tang::AstNodeFor, 66
 - Tang::AstNodeFunctionCall, 69
 - Tang::AstNodeFunctionDeclaration, 73
 - Tang::AstNodeIdentifier, 77
 - Tang::AstNodeIfElse, 83
 - Tang::AstNodeIndex, 87
 - Tang::AstNodeInteger, 92
 - Tang::AstNodeMap, 96
 - Tang::AstNodePrint, 101
 - Tang::AstNodeRangedFor, 105
 - Tang::AstNodeReturn, 109
 - Tang::AstNodeSlice, 114
 - Tang::AstNodeString, 119
 - Tang::AstNodeTernary, 123
 - Tang::AstNodeUnary, 129
 - Tang::AstNodeWhile, 133
- compileScript
 - Tang::TangBase, 324
- ComputedExpressionArray
 - Tang::ComputedExpressionArray, 150
- ComputedExpressionBoolean
 - Tang::ComputedExpressionBoolean, 163
- ComputedExpressionCompiledFunction
 - Tang::ComputedExpressionCompiledFunction, 175
- ComputedExpressionError
 - Tang::ComputedExpressionError, 189
- ComputedExpressionFloat
 - Tang::ComputedExpressionFloat, 201
- ComputedExpressionInteger
 - Tang::ComputedExpressionInteger, 216
- ComputedExpressionIterator
 - Tang::ComputedExpressionIterator, 231
- ComputedExpressionMap
 - Tang::ComputedExpressionMap, 257
- ComputedExpressionString
 - Tang::ComputedExpressionString, 271
- COPY
 - opcode.hpp, 380
- currentIndex
 - Tang::SingletonObjectPool< T >, 323
- currentRecycledIndex
 - Tang::SingletonObjectPool< T >, 323
- Default
 - Tang::AstNode, 16
 - Tang::AstNodeArray, 21
 - Tang::AstNodeAssign, 25
 - Tang::AstNodeBinary, 31
 - Tang::AstNodeBlock, 35
 - Tang::AstNodeBoolean, 39
 - Tang::AstNodeBreak, 43
 - Tang::AstNodeCast, 47
 - Tang::AstNodeContinue, 52
 - Tang::AstNodeDoWhile, 56
 - Tang::AstNodeFloat, 60
 - Tang::AstNodeFor, 64
 - Tang::AstNodeFunctionCall, 68
 - Tang::AstNodeFunctionDeclaration, 72
 - Tang::AstNodeIdentifier, 76
 - Tang::AstNodeIfElse, 81
 - Tang::AstNodeIndex, 86
 - Tang::AstNodeInteger, 91
 - Tang::AstNodeMap, 95
 - Tang::AstNodePrint, 99, 100
 - Tang::AstNodeRangedFor, 103
 - Tang::AstNodeReturn, 108
 - Tang::AstNodeSlice, 112
 - Tang::AstNodeString, 117
 - Tang::AstNodeTernary, 122
 - Tang::AstNodeUnary, 127
 - Tang::AstNodeWhile, 132
- DIVIDE
 - opcode.hpp, 380
- Divide
 - Tang::AstNodeBinary, 30
- dump
 - Tang::AstNode, 18
 - Tang::AstNodeArray, 23
 - Tang::AstNodeAssign, 27

- Tang::AstNodeBinary, [32](#)
- Tang::AstNodeBlock, [37](#)
- Tang::AstNodeBoolean, [40](#)
- Tang::AstNodeBreak, [44](#)
- Tang::AstNodeCast, [49](#)
- Tang::AstNodeContinue, [53](#)
- Tang::AstNodeDoWhile, [58](#)
- Tang::AstNodeFloat, [62](#)
- Tang::AstNodeFor, [66](#)
- Tang::AstNodeFunctionCall, [70](#)
- Tang::AstNodeFunctionDeclaration, [74](#)
- Tang::AstNodeIdentifier, [78](#)
- Tang::AstNodeIfElse, [83](#)
- Tang::AstNodeIndex, [88](#)
- Tang::AstNodeInteger, [93](#)
- Tang::AstNodeMap, [96](#)
- Tang::AstNodePrint, [101](#)
- Tang::AstNodeRangedFor, [105](#)
- Tang::AstNodeReturn, [110](#)
- Tang::AstNodeSlice, [114](#)
- Tang::AstNodeString, [119](#)
- Tang::AstNodeTernary, [124](#)
- Tang::AstNodeUnary, [129](#)
- Tang::AstNodeWhile, [134](#)
- Tang::ComputedExpression, [143](#)
- Tang::ComputedExpressionArray, [157](#)
- Tang::ComputedExpressionBoolean, [169](#)
- Tang::ComputedExpressionCompiledFunction, [182](#)
- Tang::ComputedExpressionError, [195](#)
- Tang::ComputedExpressionFloat, [210](#)
- Tang::ComputedExpressionInteger, [226](#)
- Tang::ComputedExpressionIterator, [239](#)
- Tang::ComputedExpressionIteratorEnd, [251](#)
- Tang::ComputedExpressionMap, [265](#)
- Tang::ComputedExpressionString, [280](#)
- dumpBytecode
 - Tang::Program, [316](#)
- DUMPPROGRAMCHECK
 - program-dumpBytecode.cpp, [419](#)
- EQ
 - opcode.hpp, [380](#)
- Equal
 - Tang::AstNodeBinary, [30](#)
- Error
 - Tang::Error, [285](#)
- error.cpp
 - operator<<, [418](#)
- execute
 - Tang::Program, [316](#)
- EXECUTEPROGRAMCHECK
 - program-execute.cpp, [421](#)
- FLOAT
 - opcode.hpp, [380](#)
- Float
 - Tang::AstNodeCast, [48](#)
- FUNCTION
 - opcode.hpp, [380](#)
- functionsDeclared
 - Tang::Program, [320](#)
- GarbageCollected
 - Tang::GarbageCollected, [289](#)
- get
 - Tang::SingletonObjectPool< T >, [322](#)
- get_next_token
 - Tang::HtmlEscape, [305](#)
 - Tang::HtmlEscapeAscii, [307](#)
 - Tang::TangScanner, [327](#)
 - Tang::Unescape, [329](#)
- getAst
 - Tang::Program, [316](#)
- getBytecode
 - Tang::Program, [316](#)
- getCode
 - Tang::Program, [317](#)
- getCollection
 - Tang::AstNodeIndex, [88](#)
- getIdentifiers
 - Tang::Program, [317](#)
- getIdentifiersAssigned
 - Tang::Program, [317](#)
- getIndex
 - Tang::AstNodeIndex, [88](#)
- getInstance
 - Tang::SingletonObjectPool< T >, [322](#)
- GETITERATOR
 - opcode.hpp, [381](#)
- getResult
 - Tang::Program, [317](#)
- getStrings
 - Tang::Program, [318](#)
- getValue
 - Tang::ComputedExpressionFloat, [211](#)
 - Tang::ComputedExpressionInteger, [226](#)
- GreaterThan
 - Tang::AstNodeBinary, [30](#)
- GreaterThanEqual
 - Tang::AstNodeBinary, [30](#)
- GT
 - opcode.hpp, [380](#)
- GTE
 - opcode.hpp, [380](#)
- HtmlEscape
 - Tang::HtmlEscape, [304](#)
- htmlEscape
 - unicodeString.hpp, [388](#)
- HtmlEscapeAscii
 - Tang::HtmlEscapeAscii, [306](#)
- htmlEscapeAscii
 - unicodeString.hpp, [389](#)
- include/astNode.hpp, [337](#)
- include/astNodeArray.hpp, [338](#)
- include/astNodeAssign.hpp, [339](#)
- include/astNodeBinary.hpp, [340](#)

include/astNodeBlock.hpp, 341
 include/astNodeBoolean.hpp, 342
 include/astNodeBreak.hpp, 343
 include/astNodeCast.hpp, 344
 include/astNodeContinue.hpp, 345
 include/astNodeDoWhile.hpp, 346
 include/astNodeFloat.hpp, 347
 include/astNodeFor.hpp, 348
 include/astNodeFunctionCall.hpp, 349
 include/astNodeFunctionDeclaration.hpp, 350
 include/astNodeIdentifier.hpp, 351
 include/astNodeIfElse.hpp, 352
 include/astNodeIndex.hpp, 353
 include/astNodeInteger.hpp, 354
 include/astNodeMap.hpp, 355
 include/astNodePrint.hpp, 356
 include/astNodeRangedFor.hpp, 357
 include/astNodeReturn.hpp, 358
 include/astNodeSlice.hpp, 359
 include/astNodeString.hpp, 360
 include/astNodeTernary.hpp, 361
 include/astNodeUnary.hpp, 362
 include/astNodeWhile.hpp, 363
 include/computedExpression.hpp, 364
 include/computedExpressionArray.hpp, 365
 include/computedExpressionBoolean.hpp, 366
 include/computedExpressionCompiledFunction.hpp,
 367
 include/computedExpressionError.hpp, 368
 include/computedExpressionFloat.hpp, 369
 include/computedExpressionInteger.hpp, 370
 include/computedExpressionIterator.hpp, 371
 include/computedExpressionIteratorEnd.hpp, 372
 include/computedExpressionMap.hpp, 373
 include/computedExpressionString.hpp, 374
 include/error.hpp, 375
 include/garbageCollected.hpp, 376
 include/htmlEscape.hpp, 377
 include/htmlEscapeAscii.hpp, 378
 include/macros.hpp, 379
 include/opcode.hpp, 379
 include/program.hpp, 381
 include/singletonObjectPool.hpp, 382
 include/tang.hpp, 383
 include/tangBase.hpp, 384
 include/tangScanner.hpp, 385
 include/unescape.hpp, 386
 include/unicodeString.hpp, 387
 INDEX
 opcode.hpp, 380
 INTEGER
 opcode.hpp, 380
 Integer
 Tang::AstNodeCast, 48
 is_equal
 Tang::ComputedExpression, 144–146
 Tang::ComputedExpressionArray, 157–159
 Tang::ComputedExpressionBoolean, 170–172

Tang::ComputedExpressionCompiledFunction,
 182–184
 Tang::ComputedExpressionError, 196–198
 Tang::ComputedExpressionFloat, 211–213
 Tang::ComputedExpressionInteger, 226–228
 Tang::ComputedExpressionIterator, 239, 241, 242
 Tang::ComputedExpressionIteratorEnd, 252–254
 Tang::ComputedExpressionMap, 265, 267, 268
 Tang::ComputedExpressionString, 281–283
 IsAssignment
 Tang::AstNode, 16
 Tang::AstNodeArray, 21
 Tang::AstNodeAssign, 25
 Tang::AstNodeBinary, 31
 Tang::AstNodeBlock, 35
 Tang::AstNodeBoolean, 39
 Tang::AstNodeBreak, 43
 Tang::AstNodeCast, 47
 Tang::AstNodeContinue, 52
 Tang::AstNodeDoWhile, 56
 Tang::AstNodeFloat, 60
 Tang::AstNodeFor, 64
 Tang::AstNodeFunctionCall, 68
 Tang::AstNodeFunctionDeclaration, 72
 Tang::AstNodeIdentifier, 76
 Tang::AstNodeIfElse, 81
 Tang::AstNodeIndex, 86
 Tang::AstNodeInteger, 91
 Tang::AstNodeMap, 95
 Tang::AstNodePrint, 99
 Tang::AstNodeRangedFor, 103
 Tang::AstNodeReturn, 108
 Tang::AstNodeSlice, 112
 Tang::AstNodeString, 117
 Tang::AstNodeTernary, 122
 Tang::AstNodeUnary, 127
 Tang::AstNodeWhile, 132
 isCopyNeeded
 Tang::ComputedExpression, 146
 Tang::ComputedExpressionArray, 160
 Tang::ComputedExpressionBoolean, 172
 Tang::ComputedExpressionCompiledFunction, 185
 Tang::ComputedExpressionError, 198
 Tang::ComputedExpressionFloat, 213
 Tang::ComputedExpressionInteger, 228
 Tang::ComputedExpressionIterator, 242
 Tang::ComputedExpressionIteratorEnd, 254
 Tang::ComputedExpressionMap, 268
 Tang::ComputedExpressionString, 283
 Tang::GarbageCollected, 290
 ISITERATOREND
 opcode.hpp, 381
 ITERATORNEXT
 opcode.hpp, 381
 JMP
 opcode.hpp, 380
 JMPF
 opcode.hpp, 380

- JMPF_POP
 - opcode.hpp, 380
- JMPT
 - opcode.hpp, 380
- JMPT_POP
 - opcode.hpp, 380
- length
 - Tang::UnicodeString, 331
- LessThan
 - Tang::AstNodeBinary, 30
- LessThanEqual
 - Tang::AstNodeBinary, 30
- location.hh
 - operator<<, 336, 337
- LT
 - opcode.hpp, 380
- LTE
 - opcode.hpp, 380
- make
 - Tang::GarbageCollected, 290
- makeCopy
 - Tang::ComputedExpression, 146
 - Tang::ComputedExpressionArray, 160
 - Tang::ComputedExpressionBoolean, 172
 - Tang::ComputedExpressionCompiledFunction, 185
 - Tang::ComputedExpressionError, 198
 - Tang::ComputedExpressionFloat, 213
 - Tang::ComputedExpressionInteger, 229
 - Tang::ComputedExpressionIterator, 242
 - Tang::ComputedExpressionIteratorEnd, 254
 - Tang::ComputedExpressionMap, 268
 - Tang::ComputedExpressionString, 283
 - Tang::GarbageCollected, 291
- MAP
 - opcode.hpp, 380
- MODULO
 - opcode.hpp, 380
- Modulo
 - Tang::AstNodeBinary, 30
- MULTIPLY
 - opcode.hpp, 380
- Multiply
 - Tang::AstNodeBinary, 30
- NEGATIVE
 - opcode.hpp, 380
- Negative
 - Tang::AstNodeUnary, 127
- NEQ
 - opcode.hpp, 380
- NOT
 - opcode.hpp, 380
- Not
 - Tang::AstNodeUnary, 127
- NotEqual
 - Tang::AstNodeBinary, 30
- NULLVAL
 - opcode.hpp, 380
- Opcode
 - opcode.hpp, 380
- opcode.hpp
 - ADD, 380
 - ARRAY, 380
 - ASSIGNINDEX, 380
 - BOOLEAN, 380
 - CALLFUNC, 381
 - CASTBOOLEAN, 381
 - CASTFLOAT, 381
 - CASTINTEGER, 381
 - CASTSTRING, 381
 - COPY, 380
 - DIVIDE, 380
 - EQ, 380
 - FLOAT, 380
 - FUNCTION, 380
 - GETITERATOR, 381
 - GT, 380
 - GTE, 380
 - INDEX, 380
 - INTEGER, 380
 - ISITERATOREND, 381
 - ITERATORNEXT, 381
 - JMP, 380
 - JMPF, 380
 - JMPF_POP, 380
 - JMPT, 380
 - JMPT_POP, 380
 - LT, 380
 - LTE, 380
 - MAP, 380
 - MODULO, 380
 - MULTIPLY, 380
 - NEGATIVE, 380
 - NEQ, 380
 - NOT, 380
 - NULLVAL, 380
 - Opcode, 380
 - PEEK, 380
 - POKE, 380
 - POP, 380
 - PRINT, 381
 - RETURN, 381
 - SLICE, 380
 - STRING, 380
 - SUBTRACT, 380
- Operation
 - Tang::AstNodeBinary, 30
- Operator
 - Tang::AstNodeUnary, 126
- operator std::string
 - Tang::UnicodeString, 332
- operator!
 - Tang::GarbageCollected, 291
- operator!=
 - Tang::GarbageCollected, 292

- operator<
 - Tang::GarbageCollected, 296
 - Tang::UnicodeString, 333
- operator<<
 - error.cpp, 418
 - location.hh, 336, 337
 - Tang::Error, 286
 - Tang::GarbageCollected, 303
- operator<=
 - Tang::GarbageCollected, 297
- operator>
 - Tang::GarbageCollected, 302
- operator>=
 - Tang::GarbageCollected, 302
- operator*
 - Tang::GarbageCollected, 293
- operator+
 - Tang::GarbageCollected, 294
 - Tang::UnicodeString, 332
- operator-
 - Tang::GarbageCollected, 294, 295
- operator->
 - Tang::GarbageCollected, 295
- operator/
 - Tang::GarbageCollected, 296
- operator=
 - Tang::GarbageCollected, 297
- operator==
 - Tang::GarbageCollected, 299–301
 - Tang::UnicodeString, 333
- operator%
 - Tang::GarbageCollected, 292
- Or
 - Tang::AstNodeBinary, 30
- PEEK
 - opcode.hpp, 380
- POKE
 - opcode.hpp, 380
- POP
 - opcode.hpp, 380
- popBreakStack
 - Tang::Program, 318
- popContinueStack
 - Tang::Program, 318
- PreprocessState
 - Tang::AstNode, 16
 - Tang::AstNodeArray, 21
 - Tang::AstNodeAssign, 25
 - Tang::AstNodeBinary, 31
 - Tang::AstNodeBlock, 35
 - Tang::AstNodeBoolean, 39
 - Tang::AstNodeBreak, 43
 - Tang::AstNodeCast, 47
 - Tang::AstNodeContinue, 52
 - Tang::AstNodeDoWhile, 56
 - Tang::AstNodeFloat, 60
 - Tang::AstNodeFor, 64
 - Tang::AstNodeFunctionCall, 68
 - Tang::AstNodeFunctionDeclaration, 72
 - Tang::AstNodeIdentifier, 76
 - Tang::AstNodeIfElse, 81
 - Tang::AstNodeIndex, 86
 - Tang::AstNodeInteger, 91
 - Tang::AstNodeMap, 94
 - Tang::AstNodePrint, 99
 - Tang::AstNodeRangedFor, 103
 - Tang::AstNodeReturn, 108
 - Tang::AstNodeSlice, 112
 - Tang::AstNodeString, 117
 - Tang::AstNodeTernary, 122
 - Tang::AstNodeUnary, 127
 - Tang::AstNodeWhile, 132
- PRINT
 - opcode.hpp, 381
- Program
 - Tang::Program, 313
- program-dumpBytecode.cpp
 - DUMPPROGRAMCHECK, 419
- program-execute.cpp
 - EXECUTEPROGRAMCHECK, 421
 - STACKCHECK, 421
- pushEnvironment
 - Tang::Program, 319
- recycle
 - Tang::SingletonObjectPool< T >, 323
- RETURN
 - opcode.hpp, 381
- Script
 - Tang::Program, 313
- setFunctionStackDeclaration
 - Tang::Program, 319
- setJumpTarget
 - Tang::Program, 320
- SLICE
 - opcode.hpp, 380
- src/astNode.cpp, 390
- src/astNodeArray.cpp, 391
- src/astNodeAssign.cpp, 391
- src/astNodeBinary.cpp, 392
- src/astNodeBlock.cpp, 393
- src/astNodeBoolean.cpp, 393
- src/astNodeBreak.cpp, 394
- src/astNodeCast.cpp, 395
- src/astNodeContinue.cpp, 395
- src/astNodeDoWhile.cpp, 396
- src/astNodeFloat.cpp, 397
- src/astNodeFor.cpp, 398
- src/astNodeFunctionCall.cpp, 398
- src/astNodeFunctionDeclaration.cpp, 399
- src/astNodeIdentifier.cpp, 400
- src/astNodeIfElse.cpp, 401
- src/astNodeIndex.cpp, 401
- src/astNodeInteger.cpp, 402
- src/astNodeMap.cpp, 403
- src/astNodePrint.cpp, 403

- src/astNodeRangedFor.cpp, 404
- src/astNodeReturn.cpp, 405
- src/astNodeSlice.cpp, 406
- src/astNodeString.cpp, 407
- src/astNodeTernary.cpp, 408
- src/astNodeUnary.cpp, 409
- src/astNodeWhile.cpp, 409
- src/computedExpression.cpp, 410
- src/computedExpressionArray.cpp, 411
- src/computedExpressionBoolean.cpp, 412
- src/computedExpressionCompiledFunction.cpp, 412
- src/computedExpressionError.cpp, 413
- src/computedExpressionFloat.cpp, 414
- src/computedExpressionInteger.cpp, 414
- src/computedExpressionIterator.cpp, 415
- src/computedExpressionIteratorEnd.cpp, 416
- src/computedExpressionMap.cpp, 416
- src/computedExpressionString.cpp, 417
- src/error.cpp, 417
- src/program-dumpBytecode.cpp, 419
- src/program-execute.cpp, 420
- src/program.cpp, 421
- src/tangBase.cpp, 422
- src/unicodeString.cpp, 423
- STACKCHECK
 - program-execute.cpp, 421
- STRING
 - opcode.hpp, 380
- String
 - Tang::AstNodeCast, 48
- substr
 - Tang::UnicodeString, 333
- SUBTRACT
 - opcode.hpp, 380
- Subtract
 - Tang::AstNodeBinary, 30
- 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, 30
 - And, 30
 - AstNodeBinary, 31
 - compile, 31
 - compilePreprocess, 32
 - Default, 31
 - Divide, 30
 - dump, 32
 - Equal, 30
 - GreaterThan, 30
 - GreaterThanEqual, 30
 - IsAssignment, 31
 - LessThan, 30
 - LessThanEqual, 30
 - Modulo, 30
 - Multiply, 30
 - NotEqual, 30
 - Operation, 30
 - Or, 30
 - PreprocessState, 31
 - Subtract, 30
- Tang::AstNodeBlock, 33
 - AstNodeBlock, 35
 - compile, 36
 - compilePreprocess, 36
 - Default, 35
 - dump, 37
 - IsAssignment, 35
 - PreprocessState, 35
- Tang::AstNodeBoolean, 37
 - AstNodeBoolean, 39
 - compile, 40
 - compilePreprocess, 40
 - Default, 39
 - dump, 40
 - IsAssignment, 39
 - PreprocessState, 39
- Tang::AstNodeBreak, 41
 - AstNodeBreak, 43
 - compile, 43
 - compilePreprocess, 44
 - Default, 43
 - dump, 44
 - IsAssignment, 43
 - PreprocessState, 43
- Tang::AstNodeCast, 45
 - AstNodeCast, 48
 - Boolean, 48
 - compile, 48
 - compilePreprocess, 49
 - Default, 47
 - dump, 49
 - Float, 48
 - Integer, 48
 - IsAssignment, 47

- PreprocessState, 47
- String, 48
- Type, 47
- Tang::AstNodeContinue, 50
 - AstNodeContinue, 52
 - compile, 52
 - compilePreprocess, 53
 - Default, 52
 - dump, 53
 - IsAssignment, 52
 - PreprocessState, 52
- Tang::AstNodeDoWhile, 54
 - AstNodeDoWhile, 56
 - compile, 57
 - compilePreprocess, 57
 - Default, 56
 - dump, 58
 - IsAssignment, 56
 - PreprocessState, 56
- Tang::AstNodeFloat, 58
 - AstNodeFloat, 60
 - compile, 61
 - compilePreprocess, 61
 - Default, 60
 - dump, 62
 - IsAssignment, 60
 - PreprocessState, 60
- Tang::AstNodeFor, 62
 - AstNodeFor, 65
 - compile, 65
 - compilePreprocess, 66
 - Default, 64
 - dump, 66
 - IsAssignment, 64
 - PreprocessState, 64
- Tang::AstNodeFunctionCall, 67
 - AstNodeFunctionCall, 69
 - compile, 69
 - compilePreprocess, 69
 - Default, 68
 - dump, 70
 - IsAssignment, 68
 - PreprocessState, 68
- Tang::AstNodeFunctionDeclaration, 70
 - AstNodeFunctionDeclaration, 72
 - compile, 73
 - compilePreprocess, 73
 - Default, 72
 - dump, 74
 - IsAssignment, 72
 - PreprocessState, 72
- Tang::AstNodeIdentifier, 74
 - AstNodeIdentifier, 77
 - compile, 77
 - compilePreprocess, 77
 - Default, 76
 - dump, 78
 - IsAssignment, 76
- PreprocessState, 76
- Tang::AstNodeIfElse, 79
 - AstNodeIfElse, 82
 - compile, 82
 - compilePreprocess, 83
 - Default, 81
 - dump, 83
 - IsAssignment, 81
 - PreprocessState, 81
- Tang::AstNodeIndex, 84
 - AstNodeIndex, 86
 - compile, 87
 - compilePreprocess, 87
 - Default, 86
 - dump, 88
 - getCollection, 88
 - getIndex, 88
 - IsAssignment, 86
 - PreprocessState, 86
- Tang::AstNodeInteger, 89
 - AstNodeInteger, 91
 - compile, 92
 - compilePreprocess, 92
 - Default, 91
 - dump, 93
 - IsAssignment, 91
 - PreprocessState, 91
- Tang::AstNodeMap, 93
 - AstNodeMap, 95
 - compile, 95
 - compilePreprocess, 96
 - Default, 95
 - dump, 96
 - IsAssignment, 95
 - PreprocessState, 94
- Tang::AstNodePrint, 97
 - AstNodePrint, 100
 - compile, 100
 - compilePreprocess, 101
 - Default, 99, 100
 - dump, 101
 - IsAssignment, 99
 - PreprocessState, 99
 - Type, 99
- Tang::AstNodeRangedFor, 102
 - AstNodeRangedFor, 103
 - compile, 104
 - compilePreprocess, 105
 - Default, 103
 - dump, 105
 - IsAssignment, 103
 - PreprocessState, 103
- Tang::AstNodeReturn, 106
 - AstNodeReturn, 108
 - compile, 109
 - compilePreprocess, 109
 - Default, 108
 - dump, 110

- IsAssignment, 108
- PreprocessState, 108
- Tang::AstNodeSlice, 110
 - AstNodeSlice, 113
 - compile, 113
 - compilePreprocess, 114
 - Default, 112
 - dump, 114
 - IsAssignment, 112
 - PreprocessState, 112
- Tang::AstNodeString, 115
 - AstNodeString, 117
 - compile, 118
 - compileLiteral, 118
 - compilePreprocess, 119
 - Default, 117
 - dump, 119
 - IsAssignment, 117
 - PreprocessState, 117
- Tang::AstNodeTernary, 120
 - AstNodeTernary, 122
 - compile, 123
 - compilePreprocess, 123
 - Default, 122
 - dump, 124
 - IsAssignment, 122
 - PreprocessState, 122
- Tang::AstNodeUnary, 124
 - AstNodeUnary, 127
 - compile, 127
 - compilePreprocess, 129
 - Default, 127
 - dump, 129
 - IsAssignment, 127
 - Negative, 127
 - Not, 127
 - Operator, 126
 - PreprocessState, 127
- Tang::AstNodeWhile, 130
 - AstNodeWhile, 132
 - compile, 133
 - compilePreprocess, 133
 - Default, 132
 - dump, 134
 - IsAssignment, 132
 - PreprocessState, 132
- Tang::ComputedExpression, 134
 - __add, 137
 - __asCode, 137
 - __assign_index, 137
 - __boolean, 138
 - __divide, 138
 - __equal, 138
 - __float, 139
 - __getIterator, 139
 - __index, 139
 - __integer, 140
 - __iteratorNext, 140
 - __lessThan, 140
 - __modulo, 141
 - __multiply, 141
 - __negative, 141
 - __not, 142
 - __slice, 142
 - __string, 143
 - __subtract, 143
 - dump, 143
 - is_equal, 144–146
 - isCopyNeeded, 146
 - makeCopy, 146
- Tang::ComputedExpressionArray, 147
 - __add, 150
 - __asCode, 150
 - __assign_index, 150
 - __boolean, 151
 - __divide, 151
 - __equal, 152
 - __float, 152
 - __getIterator, 152
 - __index, 153
 - __integer, 153
 - __iteratorNext, 153
 - __lessThan, 154
 - __modulo, 154
 - __multiply, 155
 - __negative, 155
 - __not, 155
 - __slice, 155
 - __string, 156
 - __subtract, 157
 - ComputedExpressionArray, 150
 - dump, 157
 - is_equal, 157–159
 - isCopyNeeded, 160
 - makeCopy, 160
- Tang::ComputedExpressionBoolean, 161
 - __add, 163
 - __asCode, 163
 - __assign_index, 164
 - __boolean, 164
 - __divide, 164
 - __equal, 165
 - __float, 165
 - __getIterator, 165
 - __index, 166
 - __integer, 166
 - __iteratorNext, 166
 - __lessThan, 167
 - __modulo, 167
 - __multiply, 167
 - __negative, 168
 - __not, 168
 - __slice, 168
 - __string, 169
 - __subtract, 169
 - ComputedExpressionBoolean, 163

- dump, 169
- is_equal, 170–172
- isCopyNeeded, 172
- makeCopy, 172
- Tang::ComputedExpressionCompiledFunction, 173
 - __add, 175
 - __asCode, 176
 - __assign_index, 176
 - __boolean, 176
 - __divide, 177
 - __equal, 177
 - __float, 178
 - __getIterator, 178
 - __index, 178
 - __integer, 179
 - __iteratorNext, 179
 - __lessThan, 179
 - __modulo, 180
 - __multiply, 180
 - __negative, 180
 - __not, 181
 - __slice, 181
 - __string, 182
 - __subtract, 182
- ComputedExpressionCompiledFunction, 175
- dump, 182
- is_equal, 182–184
- isCopyNeeded, 185
- makeCopy, 185
- Tang::ComputedExpressionError, 186
 - __add, 189
 - __asCode, 189
 - __assign_index, 190
 - __boolean, 190
 - __divide, 190
 - __equal, 191
 - __float, 191
 - __getIterator, 191
 - __index, 192
 - __integer, 192
 - __iteratorNext, 192
 - __lessThan, 193
 - __modulo, 193
 - __multiply, 193
 - __negative, 194
 - __not, 194
 - __slice, 194
 - __string, 195
 - __subtract, 195
- ComputedExpressionError, 189
- dump, 195
- is_equal, 196–198
- isCopyNeeded, 198
- makeCopy, 198
- Tang::ComputedExpressionFloat, 199
 - __add, 201
 - __asCode, 203
 - __assign_index, 203
 - __boolean, 204
 - __divide, 204
 - __equal, 204
 - __float, 205
 - __getIterator, 205
 - __index, 206
 - __integer, 206
 - __iteratorNext, 206
 - __lessThan, 207
 - __modulo, 207
 - __multiply, 208
 - __negative, 208
 - __not, 208
 - __slice, 209
 - __string, 209
 - __subtract, 210
- ComputedExpressionFloat, 201
- dump, 210
- getValue, 211
- is_equal, 211–213
- isCopyNeeded, 213
- makeCopy, 213
- Tang::ComputedExpressionInteger, 214
 - __add, 216
 - __asCode, 218
 - __assign_index, 218
 - __boolean, 219
 - __divide, 219
 - __equal, 219
 - __float, 220
 - __getIterator, 220
 - __index, 221
 - __integer, 221
 - __iteratorNext, 221
 - __lessThan, 222
 - __modulo, 222
 - __multiply, 223
 - __negative, 223
 - __not, 224
 - __slice, 224
 - __string, 225
 - __subtract, 225
- ComputedExpressionInteger, 216
- dump, 226
- getValue, 226
- is_equal, 226–228
- isCopyNeeded, 228
- makeCopy, 229
- Tang::ComputedExpressionIterator, 229
 - __add, 232
 - __asCode, 232
 - __assign_index, 232
 - __boolean, 233
 - __divide, 233
 - __equal, 234
 - __float, 234
 - __getIterator, 234
 - __index, 235

- [__integer](#), 235
 - [__iteratorNext](#), 235
 - [__lessThan](#), 236
 - [__modulo](#), 236
 - [__multiply](#), 237
 - [__negative](#), 237
 - [__not](#), 237
 - [__slice](#), 237
 - [__string](#), 238
 - [__subtract](#), 238
- [ComputedExpressionIterator](#), 231
- [dump](#), 239
- [is_equal](#), 239, 241, 242
- [isCopyNeeded](#), 242
- [makeCopy](#), 242
- [Tang::ComputedExpressionIteratorEnd](#), 243
 - [__add](#), 245
 - [__asCode](#), 245
 - [__assign_index](#), 245
 - [__boolean](#), 246
 - [__divide](#), 246
 - [__equal](#), 247
 - [__float](#), 247
 - [__getIterator](#), 247
 - [__index](#), 248
 - [__integer](#), 248
 - [__iteratorNext](#), 248
 - [__lessThan](#), 249
 - [__modulo](#), 249
 - [__multiply](#), 249
 - [__negative](#), 250
 - [__not](#), 250
 - [__slice](#), 250
 - [__string](#), 251
 - [__subtract](#), 251
- [dump](#), 251
- [is_equal](#), 252–254
- [isCopyNeeded](#), 254
- [makeCopy](#), 254
- [Tang::ComputedExpressionMap](#), 255
 - [__add](#), 258
 - [__asCode](#), 258
 - [__assign_index](#), 258
 - [__boolean](#), 259
 - [__divide](#), 259
 - [__equal](#), 260
 - [__float](#), 260
 - [__getIterator](#), 260
 - [__index](#), 261
 - [__integer](#), 261
 - [__iteratorNext](#), 261
 - [__lessThan](#), 262
 - [__modulo](#), 262
 - [__multiply](#), 263
 - [__negative](#), 263
 - [__not](#), 263
 - [__slice](#), 263
 - [__string](#), 264
 - [__subtract](#), 264
- [ComputedExpressionMap](#), 257
- [dump](#), 265
- [is_equal](#), 265, 267, 268
- [isCopyNeeded](#), 268
- [makeCopy](#), 268
- [Tang::ComputedExpressionString](#), 269
 - [__add](#), 272
 - [__asCode](#), 272
 - [__assign_index](#), 273
 - [__boolean](#), 273
 - [__divide](#), 274
 - [__equal](#), 274
 - [__float](#), 275
 - [__getIterator](#), 275
 - [__index](#), 275
 - [__integer](#), 276
 - [__iteratorNext](#), 276
 - [__lessThan](#), 277
 - [__modulo](#), 277
 - [__multiply](#), 278
 - [__negative](#), 278
 - [__not](#), 278
 - [__slice](#), 279
 - [__string](#), 280
 - [__subtract](#), 280
- [ComputedExpressionString](#), 271
- [dump](#), 280
- [is_equal](#), 281–283
- [isCopyNeeded](#), 283
- [makeCopy](#), 283
- [Tang::Error](#), 284
 - [Error](#), 285
 - [operator<<](#), 286
- [Tang::GarbageCollected](#), 286
 - [~GarbageCollected](#), 289
- [GarbageCollected](#), 289
- [isCopyNeeded](#), 290
- [make](#), 290
- [makeCopy](#), 291
- [operator!](#), 291
- [operator!=](#), 292
- [operator<](#), 296
- [operator<<](#), 303
- [operator<=](#), 297
- [operator>](#), 302
- [operator>=](#), 302
- [operator*](#), 293
- [operator+](#), 294
- [operator-](#), 294, 295
- [operator->](#), 295
- [operator/](#), 296
- [operator=](#), 297
- [operator==](#), 299–301
- [operator%](#), 292
- [Tang::HtmlEscape](#), 303
 - [get_next_token](#), 305
- [HtmlEscape](#), 304

- Tang::HtmlEscapeAscii, 305
 - get_next_token, 307
 - HtmlEscapeAscii, 306
- Tang::location, 307
- Tang::position, 309
- Tang::Program, 310
 - addBreak, 314
 - addBytecode, 314
 - addContinue, 314
 - addIdentifier, 315
 - addIdentifierAssigned, 315
 - addString, 315
 - CodeType, 313
 - dumpBytecode, 316
 - execute, 316
 - functionsDeclared, 320
 - getAst, 316
 - getBytecode, 316
 - getCode, 317
 - getIdentifiers, 317
 - getIdentifiersAssigned, 317
 - getResult, 317
 - getStrings, 318
 - popBreakStack, 318
 - popContinueStack, 318
 - Program, 313
 - pushEnvironment, 319
 - Script, 313
 - setFunctionStackDeclaration, 319
 - setJumpTarget, 320
 - Template, 313
- Tang::SingletonObjectPool< T >, 321
 - currentIndex, 323
 - currentRecycledIndex, 323
 - get, 322
 - getInstance, 322
 - recycle, 323
- Tang::TangBase, 324
 - compileScript, 324
 - TangBase, 324
- Tang::TangScanner, 325
 - get_next_token, 327
 - TangScanner, 327
- Tang::Unescape, 328
 - get_next_token, 329
 - Unescape, 329
- Tang::UnicodeString, 330
 - bytesLength, 331
 - length, 331
 - operator std::string, 332
 - operator<, 333
 - operator+, 332
 - operator==, 333
 - substr, 333
 - UnicodeString, 331
- TangBase
 - Tang::TangBase, 324
- TangScanner
 - Tang::TangScanner, 327
- Template
 - Tang::Program, 313
- test/test.cpp, 423
- test/testGarbageCollected.cpp, 425
- test/testSingletonObjectPool.cpp, 426
- test/testUnicodeString.cpp, 427
- Type
 - Tang::AstNodeCast, 47
 - Tang::AstNodePrint, 99
- Unescape
 - Tang::Unescape, 329
- unescape
 - unicodeString.hpp, 389
- UnicodeString
 - Tang::UnicodeString, 331
- unicodeString.hpp
 - htmlEscape, 388
 - htmlEscapeAscii, 389
 - unescape, 389