

Tang

0.1

Generated by Doxygen 1.9.1



<b>1 Tang: A Template Language</b>	<b>1</b>
1.1 Quick Description	1
1.2 Features	1
1.3 License	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Class Documentation</b>	<b>9</b>
5.1 Tang::AstNode Class Reference	9
5.1.1 Detailed Description	11
5.1.2 Constructor & Destructor Documentation	11
5.1.2.1 AstNode()	11
5.1.3 Member Function Documentation	11
5.1.3.1 makeCopy()	11
5.2 Tang::AstNodeBinary Class Reference	12
5.2.1 Detailed Description	14
5.2.2 Member Enumeration Documentation	14
5.2.2.1 Operation	14
5.2.3 Constructor & Destructor Documentation	14
5.2.3.1 AstNodeBinary()	14
5.2.4 Member Function Documentation	15
5.2.4.1 makeCopy()	15
5.3 Tang::AstNodeBlock Class Reference	15
5.3.1 Detailed Description	17
5.3.2 Constructor & Destructor Documentation	17
5.3.2.1 AstNodeBlock()	17
5.3.3 Member Function Documentation	17
5.3.3.1 makeCopy()	17
5.4 Tang::AstNodeBoolean Class Reference	18
5.4.1 Detailed Description	20
5.4.2 Constructor & Destructor Documentation	20
5.4.2.1 AstNodeBoolean()	20
5.4.3 Member Function Documentation	20
5.4.3.1 makeCopy()	20
5.5 Tang::AstNodeCast Class Reference	21
5.5.1 Detailed Description	23
5.5.2 Member Enumeration Documentation	23

5.5.2.1 Type . . . . .	23
5.5.3 Constructor & Destructor Documentation . . . . .	23
5.5.3.1 AstNodeCast() . . . . .	23
5.5.4 Member Function Documentation . . . . .	24
5.5.4.1 makeCopy() . . . . .	24
5.6 Tang::AstNodeFloat Class Reference . . . . .	24
5.6.1 Detailed Description . . . . .	26
5.6.2 Constructor & Destructor Documentation . . . . .	26
5.6.2.1 AstNodeFloat() . . . . .	26
5.6.3 Member Function Documentation . . . . .	26
5.6.3.1 makeCopy() . . . . .	26
5.7 Tang::AstNodeInteger Class Reference . . . . .	27
5.7.1 Detailed Description . . . . .	29
5.7.2 Constructor & Destructor Documentation . . . . .	29
5.7.2.1 AstNodeInteger() . . . . .	29
5.7.3 Member Function Documentation . . . . .	29
5.7.3.1 makeCopy() . . . . .	29
5.8 Tang::AstNodeUnary Class Reference . . . . .	30
5.8.1 Detailed Description . . . . .	32
5.8.2 Member Enumeration Documentation . . . . .	32
5.8.2.1 Operator . . . . .	32
5.8.3 Constructor & Destructor Documentation . . . . .	32
5.8.3.1 AstNodeUnary() . . . . .	32
5.8.4 Member Function Documentation . . . . .	33
5.8.4.1 makeCopy() . . . . .	33
5.9 Tang::ComputedExpression Class Reference . . . . .	33
5.9.1 Detailed Description . . . . .	34
5.9.2 Member Function Documentation . . . . .	34
5.9.2.1 __add() . . . . .	35
5.9.2.2 __boolean() . . . . .	36
5.9.2.3 __divide() . . . . .	36
5.9.2.4 __equal() . . . . .	37
5.9.2.5 __float() . . . . .	38
5.9.2.6 __integer() . . . . .	38
5.9.2.7 __lessThan() . . . . .	38
5.9.2.8 __modulo() . . . . .	39
5.9.2.9 __multiply() . . . . .	39
5.9.2.10 __negative() . . . . .	40
5.9.2.11 __not() . . . . .	40
5.9.2.12 __subtract() . . . . .	40
5.9.2.13 dump() . . . . .	41
5.9.2.14 is_equal() [1/4] . . . . .	41

5.9.2.15 <code>is_equal()</code> [2/4]	41
5.9.2.16 <code>is_equal()</code> [3/4]	42
5.9.2.17 <code>is_equal()</code> [4/4]	42
5.9.2.18 <code>makeCopy()</code>	42
5.10 Tang::ComputedExpressionBoolean Class Reference	43
5.10.1 Detailed Description	44
5.10.2 Constructor & Destructor Documentation	44
5.10.2.1 <code>ComputedExpressionBoolean()</code>	45
5.10.3 Member Function Documentation	46
5.10.3.1 <code>__add()</code>	46
5.10.3.2 <code>__boolean()</code>	46
5.10.3.3 <code>__divide()</code>	46
5.10.3.4 <code>__equal()</code>	47
5.10.3.5 <code>__float()</code>	47
5.10.3.6 <code>__integer()</code>	48
5.10.3.7 <code>__lessThan()</code>	48
5.10.3.8 <code>__modulo()</code>	48
5.10.3.9 <code>__multiply()</code>	49
5.10.3.10 <code>__negative()</code>	49
5.10.3.11 <code>__not()</code>	49
5.10.3.12 <code>__subtract()</code>	49
5.10.3.13 <code>dump()</code>	50
5.10.3.14 <code>is_equal()</code> [1/4]	50
5.10.3.15 <code>is_equal()</code> [2/4]	50
5.10.3.16 <code>is_equal()</code> [3/4]	51
5.10.3.17 <code>is_equal()</code> [4/4]	51
5.10.3.18 <code>makeCopy()</code>	52
5.11 Tang::ComputedExpressionError Class Reference	52
5.11.1 Detailed Description	54
5.11.2 Constructor & Destructor Documentation	54
5.11.2.1 <code>ComputedExpressionError()</code>	54
5.11.3 Member Function Documentation	54
5.11.3.1 <code>__add()</code>	54
5.11.3.2 <code>__boolean()</code>	55
5.11.3.3 <code>__divide()</code>	55
5.11.3.4 <code>__equal()</code>	55
5.11.3.5 <code>__float()</code>	56
5.11.3.6 <code>__integer()</code>	56
5.11.3.7 <code>__lessThan()</code>	56
5.11.3.8 <code>__modulo()</code>	57
5.11.3.9 <code>__multiply()</code>	57
5.11.3.10 <code>__negative()</code>	57

5.11.3.11 __not()	58
5.11.3.12 __subtract()	58
5.11.3.13 dump()	58
5.11.3.14 is_equal() [1/4]	59
5.11.3.15 is_equal() [2/4]	59
5.11.3.16 is_equal() [3/4]	59
5.11.3.17 is_equal() [4/4]	60
5.11.3.18 makeCopy()	60
5.12 Tang::ComputedExpressionFloat Class Reference	61
5.12.1 Detailed Description	62
5.12.2 Constructor & Destructor Documentation	62
5.12.2.1 ComputedExpressionFloat()	62
5.12.3 Member Function Documentation	63
5.12.3.1 __add()	63
5.12.3.2 __boolean()	63
5.12.3.3 __divide()	63
5.12.3.4 __equal()	64
5.12.3.5 __float()	64
5.12.3.6 __integer()	65
5.12.3.7 __lessThan()	65
5.12.3.8 __modulo()	65
5.12.3.9 __multiply()	66
5.12.3.10 __negative()	66
5.12.3.11 __not()	66
5.12.3.12 __subtract()	66
5.12.3.13 dump()	67
5.12.3.14 is_equal() [1/4]	67
5.12.3.15 is_equal() [2/4]	67
5.12.3.16 is_equal() [3/4]	68
5.12.3.17 is_equal() [4/4]	68
5.12.3.18 makeCopy()	69
5.13 Tang::ComputedExpressionInteger Class Reference	69
5.13.1 Detailed Description	71
5.13.2 Constructor & Destructor Documentation	71
5.13.2.1 ComputedExpressionInteger()	71
5.13.3 Member Function Documentation	71
5.13.3.1 __add()	71
5.13.3.2 __boolean()	72
5.13.3.3 __divide()	72
5.13.3.4 __equal()	72
5.13.3.5 __float()	73
5.13.3.6 __integer()	73

5.13.3.7 __lessThan()	73
5.13.3.8 __modulo()	74
5.13.3.9 __multiply()	74
5.13.3.10 __negative()	75
5.13.3.11 __not()	75
5.13.3.12 __subtract()	75
5.13.3.13 dump()	76
5.13.3.14 is_equal() [1/4]	76
5.13.3.15 is_equal() [2/4]	76
5.13.3.16 is_equal() [3/4]	77
5.13.3.17 is_equal() [4/4]	77
5.13.3.18 makeCopy()	77
5.14 Tang::Error Class Reference	78
5.14.1 Detailed Description	80
5.14.2 Constructor & Destructor Documentation	80
5.14.2.1 Error() [1/2]	80
5.14.2.2 Error() [2/2]	80
5.14.3 Friends And Related Function Documentation	80
5.14.3.1 operator<<	81
5.15 Tang::GarbageCollected Class Reference	81
5.15.1 Detailed Description	83
5.15.2 Constructor & Destructor Documentation	83
5.15.2.1 GarbageCollected() [1/3]	83
5.15.2.2 GarbageCollected() [2/3]	84
5.15.2.3 ~GarbageCollected()	84
5.15.2.4 GarbageCollected() [3/3]	84
5.15.3 Member Function Documentation	84
5.15.3.1 make()	84
5.15.3.2 operator"!()	85
5.15.3.3 operator"!=()	85
5.15.3.4 operator%()	86
5.15.3.5 operator*() [1/2]	87
5.15.3.6 operator*() [2/2]	87
5.15.3.7 operator+()	87
5.15.3.8 operator-() [1/2]	88
5.15.3.9 operator-() [2/2]	88
5.15.3.10 operator->()	89
5.15.3.11 operator/()	89
5.15.3.12 operator<()	90
5.15.3.13 operator<=()	90
5.15.3.14 operator=() [1/2]	91
5.15.3.15 operator=() [2/2]	91

5.15.3.16 operator==( ) [ 1/5 ] . . . . .	92
5.15.3.17 operator==( ) [ 2/5 ] . . . . .	92
5.15.3.18 operator==( ) [ 3/5 ] . . . . .	93
5.15.3.19 operator==( ) [ 4/5 ] . . . . .	93
5.15.3.20 operator==( ) [ 5/5 ] . . . . .	93
5.15.3.21 operator>( ) . . . . .	94
5.15.3.22 operator>=( ) . . . . .	94
5.15.4 Friends And Related Function Documentation . . . . .	95
5.15.4.1 operator<< . . . . .	95
5.16 Tang::location Class Reference . . . . .	95
5.16.1 Detailed Description . . . . .	97
5.17 Tang::position Class Reference . . . . .	97
5.17.1 Detailed Description . . . . .	98
5.18 Tang::Program Class Reference . . . . .	98
5.18.1 Detailed Description . . . . .	100
5.18.2 Member Enumeration Documentation . . . . .	100
5.18.2.1 CodeType . . . . .	100
5.18.3 Constructor & Destructor Documentation . . . . .	100
5.18.3.1 Program( ) . . . . .	100
5.18.4 Member Function Documentation . . . . .	100
5.18.4.1 addBytecode( ) . . . . .	101
5.18.4.2 dumpBytecode( ) . . . . .	101
5.18.4.3 execute( ) . . . . .	101
5.18.4.4 getAst( ) . . . . .	101
5.18.4.5 getCode( ) . . . . .	102
5.18.4.6 getResult( ) . . . . .	102
5.19 Tang::SingletonObjectPool< T > Class Template Reference . . . . .	102
5.19.1 Detailed Description . . . . .	103
5.19.2 Member Function Documentation . . . . .	103
5.19.2.1 get( ) . . . . .	103
5.19.2.2 getInstance( ) . . . . .	103
5.19.2.3 recycle( ) . . . . .	103
5.20 Tang::TangBase Class Reference . . . . .	104
5.20.1 Detailed Description . . . . .	104
5.20.2 Constructor & Destructor Documentation . . . . .	104
5.20.2.1 TangBase( ) . . . . .	104
5.20.3 Member Function Documentation . . . . .	104
5.20.3.1 compileScript( ) . . . . .	104
5.21 Tang::TangScanner Class Reference . . . . .	105
5.21.1 Detailed Description . . . . .	106
5.21.2 Constructor & Destructor Documentation . . . . .	106
5.21.2.1 TangScanner( ) . . . . .	106



5.21.3 Member Function Documentation	106
5.21.3.1 get_next_token()	106
<b>6 File Documentation</b>	<b>109</b>
6.1 build/generated/location.hh File Reference	109
6.1.1 Detailed Description	110
6.1.2 Function Documentation	110
6.1.2.1 operator<<() [1/2]	110
6.1.2.2 operator<<() [2/2]	111
6.2 include/astNode.hpp File Reference	111
6.2.1 Detailed Description	112
6.3 include/astNodeBinary.hpp File Reference	112
6.3.1 Detailed Description	113
6.4 include/astNodeBlock.hpp File Reference	113
6.4.1 Detailed Description	114
6.5 include/astNodeBoolean.hpp File Reference	114
6.5.1 Detailed Description	115
6.6 include/astNodeCast.hpp File Reference	115
6.6.1 Detailed Description	116
6.7 include/astNodeFloat.hpp File Reference	116
6.7.1 Detailed Description	117
6.8 include/astNodeInteger.hpp File Reference	117
6.8.1 Detailed Description	118
6.9 include/astNodeUnary.hpp File Reference	118
6.9.1 Detailed Description	119
6.10 include/computedExpression.hpp File Reference	119
6.10.1 Detailed Description	120
6.11 include/computedExpressionBoolean.hpp File Reference	120
6.11.1 Detailed Description	121
6.12 include/computedExpressionError.hpp File Reference	121
6.12.1 Detailed Description	122
6.13 include/computedExpressionFloat.hpp File Reference	122
6.13.1 Detailed Description	123
6.14 include/computedExpressionInteger.hpp File Reference	123
6.14.1 Detailed Description	124
6.15 include/error.hpp File Reference	124
6.15.1 Detailed Description	125
6.16 include/garbageCollected.hpp File Reference	125
6.16.1 Detailed Description	125
6.17 include/macros.hpp File Reference	126
6.17.1 Detailed Description	126
6.17.2 Macro Definition Documentation	126

6.17.2.1 TANG_UNUSED	126
6.18 include/opcode.hpp File Reference	126
6.18.1 Detailed Description	127
6.18.2 Enumeration Type Documentation	127
6.18.2.1 Opcode	127
6.19 include/program.hpp File Reference	128
6.19.1 Detailed Description	128
6.20 include/singletonObjectPool.hpp File Reference	129
6.20.1 Detailed Description	129
6.21 include/tang.hpp File Reference	130
6.21.1 Detailed Description	130
6.22 include/tangBase.hpp File Reference	131
6.22.1 Detailed Description	132
6.23 include/tangScanner.hpp File Reference	132
6.23.1 Detailed Description	133
6.24 src/astNode.cpp File Reference	133
6.24.1 Detailed Description	133
6.25 src/astNodeBinary.cpp File Reference	134
6.25.1 Detailed Description	134
6.26 src/astNodeBlock.cpp File Reference	134
6.26.1 Detailed Description	135
6.27 src/astNodeBoolean.cpp File Reference	135
6.27.1 Detailed Description	136
6.28 src/astNodeCast.cpp File Reference	136
6.28.1 Detailed Description	137
6.29 src/astNodeFloat.cpp File Reference	137
6.29.1 Detailed Description	138
6.30 src/astNodeInteger.cpp File Reference	138
6.30.1 Detailed Description	139
6.31 src/astNodeUnary.cpp File Reference	139
6.31.1 Detailed Description	140
6.32 src/computedExpression.cpp File Reference	140
6.32.1 Detailed Description	141
6.33 src/computedExpressionBoolean.cpp File Reference	141
6.33.1 Detailed Description	142
6.34 src/computedExpressionError.cpp File Reference	142
6.34.1 Detailed Description	142
6.35 src/computedExpressionFloat.cpp File Reference	142
6.35.1 Detailed Description	143
6.36 src/computedExpressionInteger.cpp File Reference	143
6.36.1 Detailed Description	144
6.37 src/error.cpp File Reference	144

---

6.37.1 Detailed Description . . . . .	144
6.37.2 Function Documentation . . . . .	144
6.37.2.1 operator<<() . . . . .	144
6.38 src/program-dumpBytecode.cpp File Reference . . . . .	145
6.38.1 Detailed Description . . . . .	145
6.38.2 Macro Definition Documentation . . . . .	146
6.38.2.1 DUMPPROGRAMCHECK . . . . .	146
6.39 src/program-execute.cpp File Reference . . . . .	146
6.39.1 Detailed Description . . . . .	147
6.39.2 Macro Definition Documentation . . . . .	147
6.39.2.1 EXECUTEPROGRAMCHECK . . . . .	147
6.39.2.2 STACKCHECK . . . . .	147
6.40 src/program.cpp File Reference . . . . .	148
6.40.1 Detailed Description . . . . .	148
6.41 src/tangBase.cpp File Reference . . . . .	148
6.41.1 Detailed Description . . . . .	149
6.42 test/test.cpp File Reference . . . . .	149
6.42.1 Detailed Description . . . . .	150
6.43 test/testGarbageCollected.cpp File Reference . . . . .	151
6.43.1 Detailed Description . . . . .	151
6.44 test/testSingletonObjectPool.cpp File Reference . . . . .	151
6.44.1 Detailed Description . . . . .	152
<b>Index</b>	<b>153</b>



# 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 . . . . .	9
Tang::AstNodeBinary . . . . .	12
Tang::AstNodeBlock . . . . .	15
Tang::AstNodeBoolean . . . . .	18
Tang::AstNodeCast . . . . .	21
Tang::AstNodeFloat . . . . .	24
Tang::AstNodeInteger . . . . .	27
Tang::AstNodeUnary . . . . .	30
Tang::ComputedExpression . . . . .	33
Tang::ComputedExpressionBoolean . . . . .	43
Tang::ComputedExpressionError . . . . .	52
Tang::ComputedExpressionFloat . . . . .	61
Tang::ComputedExpressionInteger . . . . .	69
Tang::Error . . . . .	78
Tang::GarbageCollected . . . . .	81
Tang::location . . . . .	95
Tang::position . . . . .	97
Tang::Program . . . . .	98
Tang::SingletonObjectPool< T > . . . . .	102
Tang::TangBase . . . . .	104
TangTangFlexLexer	
Tang::TangScanner . . . . .	105





## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">Tang::AstNode</a>	Base class for representing nodes of an Abstract Syntax Tree (AST) . . . . .	9
<a href="#">Tang::AstNodeBinary</a>	An <a href="#">AstNode</a> that represents a binary expression . . . . .	12
<a href="#">Tang::AstNodeBlock</a>	An <a href="#">AstNode</a> that represents a code block . . . . .	15
<a href="#">Tang::AstNodeBoolean</a>	An <a href="#">AstNode</a> that represents a boolean literal . . . . .	18
<a href="#">Tang::AstNodeCast</a>	An <a href="#">AstNode</a> that represents a typecast of an expression . . . . .	21
<a href="#">Tang::AstNodeFloat</a>	An <a href="#">AstNode</a> that represents an float literal . . . . .	24
<a href="#">Tang::AstNodeInteger</a>	An <a href="#">AstNode</a> that represents an integer literal . . . . .	27
<a href="#">Tang::AstNodeUnary</a>	An <a href="#">AstNode</a> that represents a unary negation . . . . .	30
<a href="#">Tang::ComputedExpression</a>	Represents the result of a computation that has been executed . . . . .	33
<a href="#">Tang::ComputedExpressionBoolean</a>	Represents an Boolean that is the result of a computation . . . . .	43
<a href="#">Tang::ComputedExpressionError</a>	Represents a Runtime <a href="#">Error</a> . . . . .	52
<a href="#">Tang::ComputedExpressionFloat</a>	Represents a Float that is the result of a computation . . . . .	61
<a href="#">Tang::ComputedExpressionInteger</a>	Represents an Integer that is the result of a computation . . . . .	69
<a href="#">Tang::Error</a>	Used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error . . . . .	78
<a href="#">Tang::GarbageCollected</a>	A container that acts as a resource-counting garbage collector for the specified type . . . . .	81
<a href="#">Tang::location</a>	Two points in a source file . . . . .	95
<a href="#">Tang::position</a>	A point in a source file . . . . .	97

<a href="#">Tang::Program</a>	
Represents a compiled script or template that may be executed . . . . .	98
<a href="#">Tang::SingletonObjectPool&lt; T &gt;</a>	
A thread-safe, singleton object pool of the designated type . . . . .	102
<a href="#">Tang::TangBase</a>	
The base class for the Tang programming language . . . . .	104
<a href="#">Tang::TangScanner</a>	
The Flex lexer class for the main Tang language . . . . .	105

## 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 . . . . .	109
include/astNode.hpp	
Declare the <code>Tang::AstNode</code> base class . . . . .	111
include/astNodeBinary.hpp	
Declare the <code>Tang::AstNodeBinary</code> class . . . . .	112
include/astNodeBlock.hpp	
Declare the <code>Tang::AstNodeBlock</code> class . . . . .	113
include/astNodeBoolean.hpp	
Declare the <code>Tang::AstNodeBoolean</code> class . . . . .	114
include/astNodeCast.hpp	
Declare the <code>Tang::AstNodeCast</code> class . . . . .	115
include/astNodeFloat.hpp	
Declare the <code>Tang::AstNodeFloat</code> class . . . . .	116
include/astNodeInteger.hpp	
Declare the <code>Tang::AstNodeInteger</code> class . . . . .	117
include/astNodeUnary.hpp	
Declare the <code>Tang::AstNodeUnary</code> class . . . . .	118
include/computedExpression.hpp	
Declare the <code>Tang::ComputedExpression</code> base class . . . . .	119
include/computedExpressionBoolean.hpp	
Declare the <code>Tang::ComputedExpressionBoolean</code> class . . . . .	120
include/computedExpressionError.hpp	
Declare the <code>Tang::ComputedExpressionError</code> class . . . . .	121
include/computedExpressionFloat.hpp	
Declare the <code>Tang::ComputedExpressionFloat</code> class . . . . .	122
include/computedExpressionInteger.hpp	
Declare the <code>Tang::ComputedExpressionInteger</code> class . . . . .	123
include/error.hpp	
Declare the <code>Tang::Error</code> class used to describe syntax and runtime errors . . . . .	124
include/garbageCollected.hpp	
Declare the <code>Tang::GarbageCollected</code> class . . . . .	125
include/macros.hpp	
Contains generic macros . . . . .	126
include/opcode.hpp	
Declare the Opcodes used in the Bytecode representation of a program . . . . .	126

include/ <a href="#">program.hpp</a>	Declare the <a href="#">Tang::Program</a> class used to compile and execute source code . . . . .	128
include/ <a href="#">singletonObjectPool.hpp</a>	Declare the <a href="#">Tang::SingletonObjectPool</a> class . . . . .	129
include/ <a href="#">tang.hpp</a>	Header file supplied for use by 3rd party code so that they can easily include all necessary headers . . . . .	130
include/ <a href="#">tangBase.hpp</a>	Declare the <a href="#">Tang::TangBase</a> class used to interact with Tang . . . . .	131
include/ <a href="#">tangScanner.hpp</a>	Declare the <a href="#">Tang::TangScanner</a> used to tokenize a Tang script . . . . .	132
src/ <a href="#">astNode.cpp</a>	Define the <a href="#">Tang::AstNode</a> class . . . . .	133
src/ <a href="#">astNodeBinary.cpp</a>	Define the <a href="#">Tang::AstNodeBinary</a> class . . . . .	134
src/ <a href="#">astNodeBlock.cpp</a>	Define the <a href="#">Tang::AstNodeBlock</a> class . . . . .	134
src/ <a href="#">astNodeBoolean.cpp</a>	Define the <a href="#">Tang::AstNodeBoolean</a> class . . . . .	135
src/ <a href="#">astNodeCast.cpp</a>	Define the <a href="#">Tang::AstNodeCast</a> class . . . . .	136
src/ <a href="#">astNodeFloat.cpp</a>	Define the <a href="#">Tang::AstNodeFloat</a> class . . . . .	137
src/ <a href="#">astNodeInteger.cpp</a>	Define the <a href="#">Tang::AstNodeInteger</a> class . . . . .	138
src/ <a href="#">astNodeUnary.cpp</a>	Define the <a href="#">Tang::AstNodeUnary</a> class . . . . .	139
src/ <a href="#">computedExpression.cpp</a>	Define the <a href="#">Tang::ComputedExpression</a> class . . . . .	140
src/ <a href="#">computedExpressionBoolean.cpp</a>	Define the <a href="#">Tang::ComputedExpressionBoolean</a> class . . . . .	141
src/ <a href="#">computedExpressionError.cpp</a>	Define the <a href="#">Tang::ComputedExpressionError</a> class . . . . .	142
src/ <a href="#">computedExpressionFloat.cpp</a>	Define the <a href="#">Tang::ComputedExpressionFloat</a> class . . . . .	142
src/ <a href="#">computedExpressionInteger.cpp</a>	Define the <a href="#">Tang::ComputedExpressionInteger</a> class . . . . .	143
src/ <a href="#">error.cpp</a>	Define the <a href="#">Tang::Error</a> class . . . . .	144
src/ <a href="#">program-dumpBytecode.cpp</a>	Define the <a href="#">Tang::Program::dumpBytecode</a> method . . . . .	145
src/ <a href="#">program-execute.cpp</a>	Define the <a href="#">Tang::Program::execute</a> method . . . . .	146
src/ <a href="#">program.cpp</a>	Define the <a href="#">Tang::Program</a> class . . . . .	148
src/ <a href="#">tangBase.cpp</a>	Define the <a href="#">Tang::TangBase</a> class . . . . .	148
test/ <a href="#">test.cpp</a>	Test the general language behaviors . . . . .	149
test/ <a href="#">testGarbageCollected.cpp</a>	Test the generic behavior of the <a href="#">Tang::GarbageCollected</a> class . . . . .	151
test/ <a href="#">testSingletonObjectPool.cpp</a>	Test the generic behavior of the <a href="#">Tang::SingletonObjectPool</a> class . . . . .	151

## Chapter 5

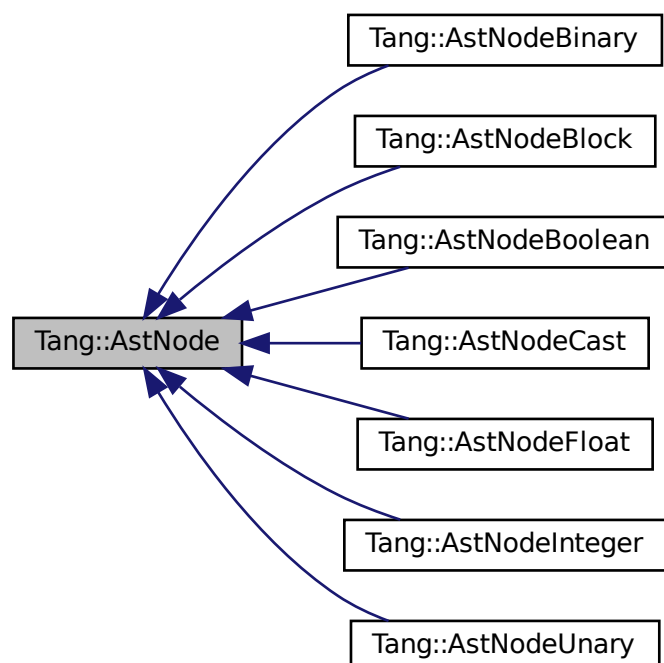
# Class Documentation

### 5.1 Tang::AstNode Class Reference

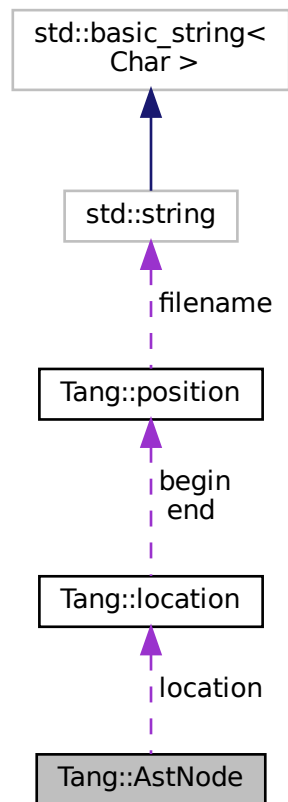
Base class for representing nodes of an Abstract Syntax Tree (AST).

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

Inheritance diagram for Tang::AstNode:



Collaboration diagram for Tang::AstNode:



## Public Member Functions

- 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 `std::shared_ptr< AstNode > makeCopy` () const  
*Provide a copy of the `AstNode` (recursively, if appropriate).*

## Protected Member Functions

- `AstNode` (`Tang::location location`)  
*The generic constructor.*

## 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).

There will be *many* derived classes, each one conveying the syntactic meaning of the code that it represents.

### 5.1.2 Constructor & Destructor Documentation

#### 5.1.2.1 AstNode()

```
AstNode::AstNode (
    Tang::location location ) [protected]
```

The generic constructor.

It should never be called on its own.

#### Parameters

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

### 5.1.3 Member Function Documentation

#### 5.1.3.1 makeCopy()

```
shared_ptr< AstNode > AstNode::makeCopy ( ) const [virtual]
```

Provide a copy of the [AstNode](#) (recursively, if appropriate).

#### Returns

A pointer to a new [AstNode](#) that is a copy of the current [AstNode](#).

Reimplemented in [Tang::AstNodeUnary](#), [Tang::AstNodeInteger](#), [Tang::AstNodeFloat](#), [Tang::AstNodeCast](#), [Tang::AstNodeBoolean](#), [Tang::AstNodeBlock](#), and [Tang::AstNodeBinary](#).

Here is the call graph for this function:



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

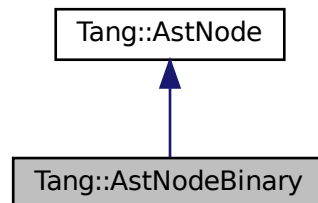
- [include/astNode.hpp](#)
- [src/astNode.cpp](#)

## 5.2 Tang::AstNodeBinary Class Reference

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

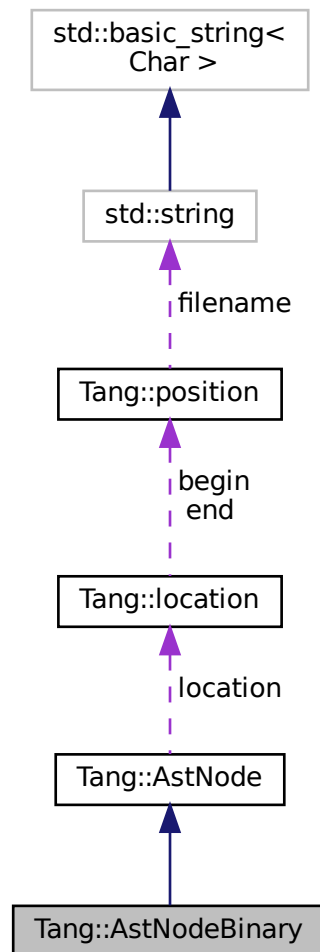
```
#include <astNodeBinary.hpp>
```

Inheritance diagram for Tang::AstNodeBinary:





Collaboration diagram for Tang::AstNodeBinary:



## Public Types

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

## Public Member Functions

- [AstNodeBinary](#) ([Operation](#) op, std::shared\_ptr< [AstNode](#) > lhs, std::shared\_ptr< [AstNode](#) > rhs, [Tang::location](#) location)  
*The constructor.*
- virtual std::string [dump](#) (std::string indent="") const override  
*Return a string that describes the contents of the node.*

- virtual void `compile` (`Tang::Program` &program) const override  
*Compile the ast of the provided `Tang::Program`.*
- virtual shared\_ptr< `AstNode` > `makeCopy` () const override  
*Provide a copy of the `AstNode` (recursively, if appropriate).*

## Protected Attributes

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

## 5.2.1 Detailed Description

An `AstNode` that represents a binary expression.

## 5.2.2 Member Enumeration Documentation

### 5.2.2.1 Operation

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

#### Enumerator

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

## 5.2.3 Constructor & Destructor Documentation

### 5.2.3.1 AstNodeBinary()

```
AstNodeBinary::AstNodeBinary (
    Operation op,
```

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

The constructor.

#### Parameters

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

## 5.2.4 Member Function Documentation

### 5.2.4.1 makeCopy()

```
shared_ptr< AstNode > AstNodeBinary::makeCopy ( ) const [override], [virtual]
```

Provide a copy of the [AstNode](#) (recursively, if appropriate).

#### Returns

A pointer to a new [AstNode](#) that is a copy of the current [AstNode](#).

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

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

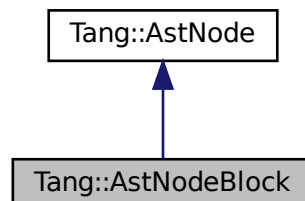
- include/[astNodeBinary.hpp](#)
- src/[astNodeBinary.cpp](#)

## 5.3 Tang::AstNodeBlock Class Reference

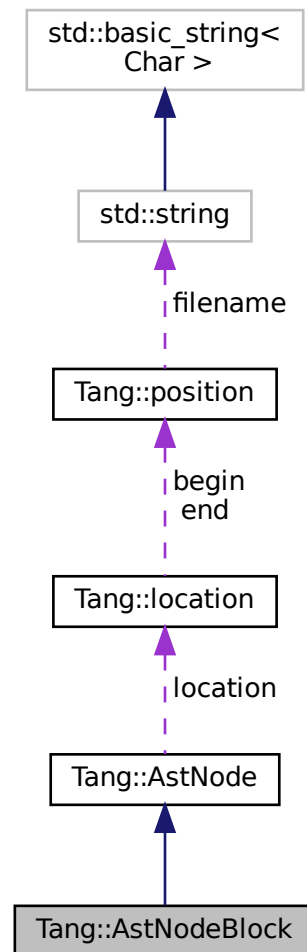
An [AstNode](#) that represents a code block.

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

Inheritance diagram for Tang::AstNodeBlock:



Collaboration diagram for Tang::AstNodeBlock:



## Public Member Functions

- [AstNodeBlock](#) (const std::vector< std::shared\_ptr< [AstNode](#) >> &statements, [Tang::location](#) location)  
*The constructor.*
- virtual std::string [dump](#) (std::string indent="") const override  
*Return a string that describes the contents of the node.*
- virtual void [compile](#) ([Tang::Program](#) &program) const override  
*Compile the ast of the provided [Tang::Program](#).*
- virtual shared\_ptr< [AstNode](#) > [makeCopy](#) () const override  
*Provide a copy of the [AstNode](#) (recursively, if appropriate).*

## Protected Attributes

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

### 5.3.1 Detailed Description

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

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 AstNodeBlock()

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

The constructor.

##### Parameters

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

### 5.3.3 Member Function Documentation

#### 5.3.3.1 makeCopy()

```
shared_ptr< AstNode > AstNodeBlock::makeCopy ( ) const [override], [virtual]
```

Provide a copy of the [AstNode](#) (recursively, if appropriate).

##### Returns

A pointer to a new [AstNode](#) that is a copy of the current [AstNode](#).

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

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

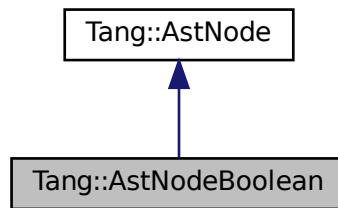
- include/[astNodeBlock.hpp](#)
- src/[astNodeBlock.cpp](#)

## 5.4 Tang::AstNodeBoolean Class Reference

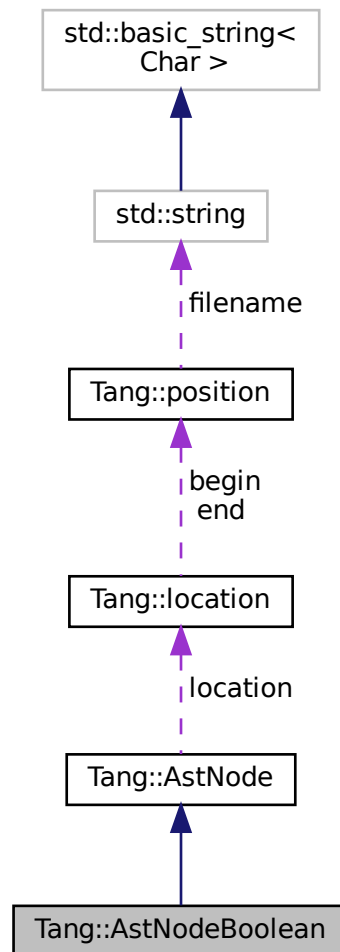
An [AstNode](#) that represents a boolean literal.

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

Inheritance diagram for Tang::AstNodeBoolean:



Collaboration diagram for Tang::AstNodeBoolean:



## Public Member Functions

- [AstNodeBoolean](#) (bool val, [Tang::location](#) location)  
*The constructor.*
- virtual `std::string` [dump](#) (std::string indent="") const override  
*Return a string that describes the contents of the node.*
- virtual void [compile](#) ([Tang::Program](#) &program) const override  
*Compile the ast of the provided [Tang::Program](#).*
- virtual `shared_ptr< AstNode >` [makeCopy](#) () const override  
*Provide a copy of the [AstNode](#) (recursively, if appropriate).*

## Protected Attributes

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

### 5.4.1 Detailed Description

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

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 AstNodeBoolean()

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

The constructor.

##### Parameters

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

### 5.4.3 Member Function Documentation

#### 5.4.3.1 makeCopy()

```
shared_ptr< AstNode > AstNodeBoolean::makeCopy ( ) const [override], [virtual]
```

Provide a copy of the [AstNode](#) (recursively, if appropriate).

##### Returns

A pointer to a new [AstNode](#) that is a copy of the current [AstNode](#).

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

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

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

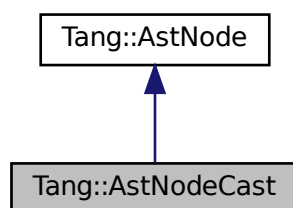


## 5.5 Tang::AstNodeCast Class Reference

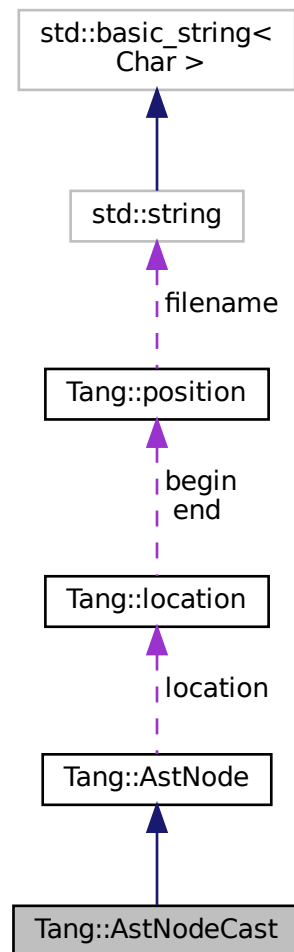
An [AstNode](#) that represents a typecast of an expression.

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

Inheritance diagram for Tang::AstNodeCast:



Collaboration diagram for Tang::AstNodeCast:



## Public Types

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

## Public Member Functions

- [AstNodeCast](#) ([Type](#) targetType, shared\_ptr< [AstNode](#) > expression, [Tang::location](#) location)
- The constructor.*
- virtual std::string [dump](#) (std::string indent="") const override
- Return a string that describes the contents of the node.*
- virtual void [compile](#) ([Tang::Program](#) &program) const override
- Compile the ast of the provided [Tang::Program](#).*
- virtual shared\_ptr< [AstNode](#) > [makeCopy](#) () const override
- Provide a copy of the [AstNode](#) (recursively, if appropriate).*

## Protected Attributes

- [Tang::location location](#)

*The location associated with this node.*

### 5.5.1 Detailed Description

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

### 5.5.2 Member Enumeration Documentation

#### 5.5.2.1 Type

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

The possible types that can be cast to.

Enumerator

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

### 5.5.3 Constructor & Destructor Documentation

#### 5.5.3.1 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.

## 5.5.4 Member Function Documentation

### 5.5.4.1 makeCopy()

```
shared_ptr< AstNode > AstNodeCast::makeCopy ( ) const [override], [virtual]
```

Provide a copy of the [AstNode](#) (recursively, if appropriate).

#### Returns

A pointer to a new [AstNode](#) that is a copy of the current [AstNode](#).

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

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

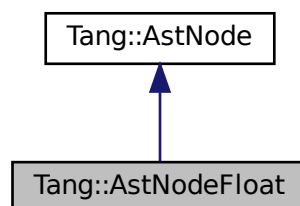
- [include/astNodeCast.hpp](#)
- [src/astNodeCast.cpp](#)

## 5.6 Tang::AstNodeFloat Class Reference

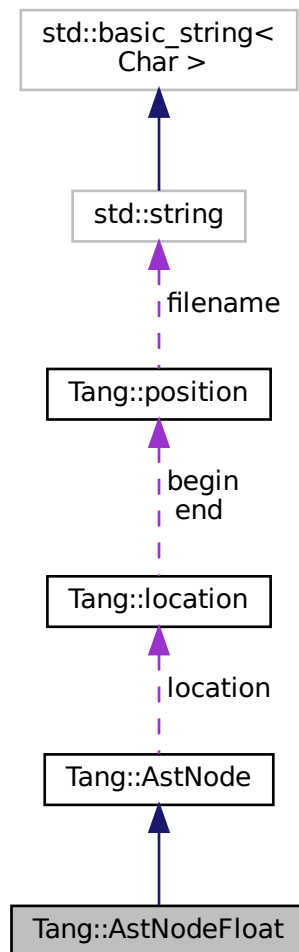
An [AstNode](#) that represents an float literal.

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

Inheritance diagram for Tang::AstNodeFloat:



Collaboration diagram for Tang::AstNodeFloat:



## Public Member Functions

- [AstNodeFloat](#) (double number, [Tang::location](#) location)  
*The constructor.*
- virtual `std::string` [dump](#) (`std::string` indent="") const override  
*Return a string that describes the contents of the node.*
- virtual void [compile](#) ([Tang::Program](#) &program) const override  
*Compile the ast of the provided [Tang::Program](#).*
- virtual `shared_ptr< AstNode >` [makeCopy](#) () const override  
*Provide a copy of the [AstNode](#) (recursively, if appropriate).*

## Protected Attributes

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

### 5.6.1 Detailed Description

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

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

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 AstNodeFloat()

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

The constructor.

##### Parameters

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

### 5.6.3 Member Function Documentation

#### 5.6.3.1 makeCopy()

```
shared_ptr< AstNode > AstNodeFloat::makeCopy ( ) const [override], [virtual]
```

Provide a copy of the [AstNode](#) (recursively, if appropriate).

##### Returns

A pointer to a new [AstNode](#) that is a copy of the current [AstNode](#).

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

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

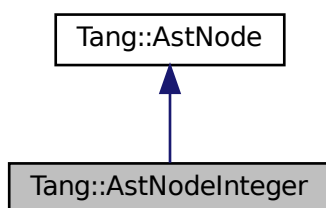
- include/[astNodeFloat.hpp](#)
- src/[astNodeFloat.cpp](#)

## 5.7 Tang::AstNodeInteger Class Reference

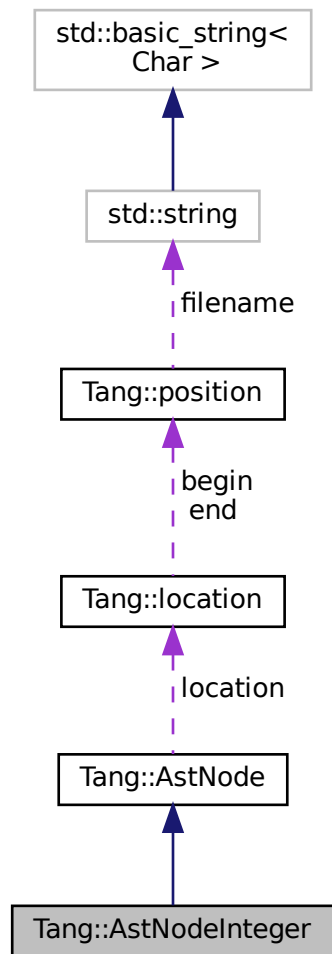
An [AstNode](#) that represents an integer literal.

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

Inheritance diagram for Tang::AstNodeInteger:



Collaboration diagram for Tang::AstNodeInteger:



## Public Member Functions

- [AstNodeInteger](#) (int64\_t number, [Tang::location](#) location)  
*The constructor.*
- virtual std::string [dump](#) (std::string indent="") const override  
*Return a string that describes the contents of the node.*
- virtual void [compile](#) ([Tang::Program](#) &program) const override  
*Compile the ast of the provided Tang::Program.*
- virtual shared\_ptr< [AstNode](#) > [makeCopy](#) () const override  
*Provide a copy of the AstNode (recursively, if appropriate).*

## Protected Attributes

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



### 5.7.1 Detailed Description

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

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

### 5.7.2 Constructor & Destructor Documentation

#### 5.7.2.1 AstNodeInteger()

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

The constructor.

##### Parameters

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

### 5.7.3 Member Function Documentation

#### 5.7.3.1 makeCopy()

```
shared_ptr< AstNode > AstNodeInteger::makeCopy ( ) const [override], [virtual]
```

Provide a copy of the [AstNode](#) (recursively, if appropriate).

##### Returns

A pointer to a new [AstNode](#) that is a copy of the current [AstNode](#).

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

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

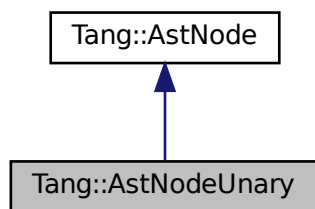
- [include/astNodeInteger.hpp](#)
- [src/astNodeInteger.cpp](#)

## 5.8 Tang::AstNodeUnary Class Reference

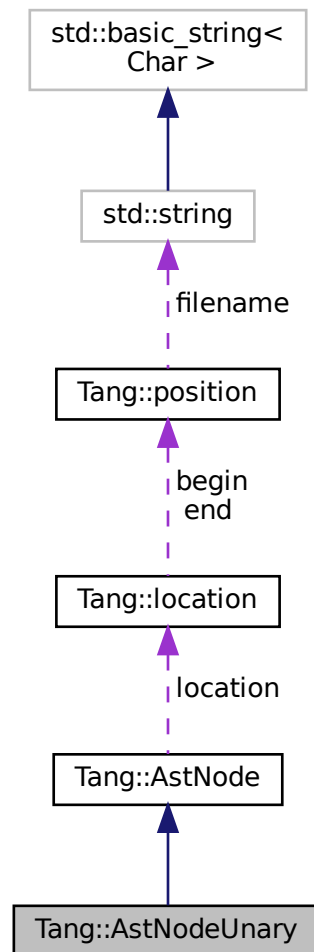
An [AstNode](#) that represents a unary negation.

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

Inheritance diagram for Tang::AstNodeUnary:



Collaboration diagram for Tang::AstNodeUnary:



## Public Types

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

## Public Member Functions

- [AstNodeUnary](#) ([Operator](#) op, shared\_ptr< [AstNode](#) > operand, [Tang::location](#) location)
- The constructor.*
- virtual std::string [dump](#) (std::string indent="") const override
- Return a string that describes the contents of the node.*
- virtual void [compile](#) ([Tang::Program](#) &program) const override
- Compile the ast of the provided [Tang::Program](#).*
- virtual shared\_ptr< [AstNode](#) > [makeCopy](#) () const override
- Provide a copy of the [AstNode](#) (recursively, if appropriate).*

## Protected Attributes

- [Tang::location location](#)

*The location associated with this node.*

### 5.8.1 Detailed Description

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

### 5.8.2 Member Enumeration Documentation

#### 5.8.2.1 Operator

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

The type of operation.

Enumerator

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

### 5.8.3 Constructor & Destructor Documentation

#### 5.8.3.1 AstNodeUnary()

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

The constructor.

Parameters

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

## 5.8.4 Member Function Documentation

### 5.8.4.1 makeCopy()

```
shared_ptr< AstNode > AstNodeUnary::makeCopy ( ) const [override], [virtual]
```

Provide a copy of the [AstNode](#) (recursively, if appropriate).

#### Returns

A pointer to a new [AstNode](#) that is a copy of the current [AstNode](#).

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

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

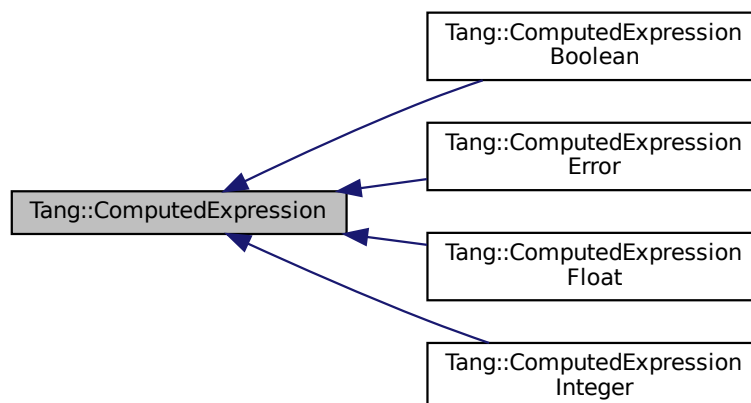
- [include/astNodeUnary.hpp](#)
- [src/astNodeUnary.cpp](#)

## 5.9 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 `ComputedExpression * makeCopy () const`  
*Make a copy of the `ComputedExpression` (recursively, if appropriate).*
- virtual `bool is_equal (const int &val) const`  
*Check whether or not the computed expression is equal to another value.*
- virtual `bool is_equal (const double &val) const`  
*Check whether or not the computed expression is equal to another value.*
- virtual `bool is_equal (const bool &val) const`  
*Check whether or not the computed expression is equal to another value.*
- virtual `bool is_equal (const Error &val) const`  
*Check whether or not the computed expression is equal to another value.*
- virtual `GarbageCollected __add (const GarbageCollected &rhs) const`  
*Compute the result of adding this value and the supplied value.*
- virtual `GarbageCollected __subtract (const GarbageCollected &rhs) const`  
*Compute the result of subtracting this value and the supplied value.*
- virtual `GarbageCollected __multiply (const GarbageCollected &rhs) const`  
*Compute the result of multiplying this value and the supplied value.*
- virtual `GarbageCollected __divide (const GarbageCollected &rhs) const`  
*Compute the result of dividing this value and the supplied value.*
- virtual `GarbageCollected __modulo (const GarbageCollected &rhs) const`  
*Compute the result of moduloing this value and the supplied value.*
- virtual `GarbageCollected __negative () const`  
*Compute the result of negating this value.*
- virtual `GarbageCollected __not () const`  
*Compute the logical not of this value.*
- virtual `GarbageCollected __lessThan (const GarbageCollected &rhs) const`  
*Compute the "less than" comparison.*
- virtual `GarbageCollected __equal (const GarbageCollected &rhs) const`  
*Perform an equality test.*
- virtual `GarbageCollected __integer () const`  
*Perform a type cast to integer.*
- virtual `GarbageCollected __float () const`  
*Perform a type cast to float.*
- virtual `GarbageCollected __boolean () const`  
*Perform a type cast to boolean.*

### 5.9.1 Detailed Description

Represents the result of a computation that has been executed.

### 5.9.2 Member Function Documentation

### 5.9.2.1 \_\_add()

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

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

**Parameters**

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

**Returns**

The result of the operation.

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

**5.9.2.2 \_\_boolean()**

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

Perform a type cast to boolean.

**Returns**

The result of the the operation.

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

**5.9.2.3 \_\_divide()**

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

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

**Parameters**

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to divide this by.
------------	---

**Returns**

The result of the operation.

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



#### 5.9.2.4 `__equal()`

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

Perform an equalit test.

## Parameters

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

## Returns

The result of the the operation.

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

**5.9.2.5 `__float()`**

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

Perform a type cast to float.

## Returns

The result of the the operation.

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

**5.9.2.6 `__integer()`**

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

Perform a type cast to integer.

## Returns

The result of the the operation.

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

**5.9.2.7 `__lessThan()`**

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

Compute the "less than" comparison.

## Parameters

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

## Returns

The result of the the operation.

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

## 5.9.2.8 \_\_modulo()

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

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

## Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to modulo this by.
------------	---

## Returns

The result of the operation.

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

## 5.9.2.9 \_\_multiply()

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

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

## Parameters

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

## Returns

The result of the operation.

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

#### 5.9.2.10 `__negative()`

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

Compute the result of negating this value.

##### Returns

The result of the operation.

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

#### 5.9.2.11 `__not()`

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

Compute the logical not of this value.

##### Returns

The result of the operation.

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

#### 5.9.2.12 `__subtract()`

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

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

##### Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to subtract from this.
------------	---

##### Returns

The result of the operation.

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

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

### 5.9.2.14 is\_equal() [1/4]

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

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

#### Parameters

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

#### Returns

True if equal, false if not.

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

### 5.9.2.15 is\_equal() [2/4]

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

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

#### Parameters

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

#### Returns

True if equal, false if not.

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

#### 5.9.2.16 `is_equal()` [3/4]

```
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](#).

#### 5.9.2.17 `is_equal()` [4/4]

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

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

##### Parameters

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

##### Returns

True if equal, false if not.

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

#### 5.9.2.18 `makeCopy()`

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

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

#### Returns

A pointer to the new [ComputedExpression](#).

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

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

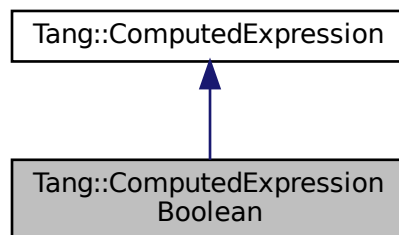
- [include/computedExpression.hpp](#)
- [src/computedExpression.cpp](#)

## 5.10 Tang::ComputedExpressionBoolean Class Reference

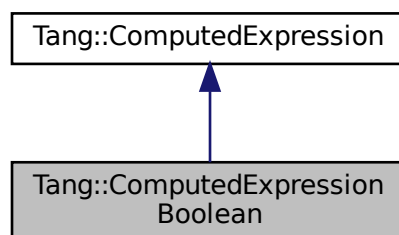
Represents an Boolean that is the result of a computation.

```
#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.*
- [ComputedExpression](#) \* [makeCopy](#) () const override  
*Make a copy of the [ComputedExpression](#) (recursively, if appropriate).*
- virtual bool [is\\_equal](#) (const bool &val) const override  
*Check whether or not the computed expression is equal to another value.*
- virtual [GarbageCollected](#) [\\_\\_not](#) () const override  
*Compute the logical not of this value.*
- virtual [GarbageCollected](#) [\\_\\_equal](#) (const [GarbageCollected](#) &rhs) const override  
*Perform an equalit test.*
- virtual [GarbageCollected](#) [\\_\\_integer](#) () const override  
*Perform a type cast to integer.*
- virtual [GarbageCollected](#) [\\_\\_float](#) () const override  
*Perform a type cast to float.*
- virtual [GarbageCollected](#) [\\_\\_boolean](#) () const override  
*Perform a type cast to boolean.*
- virtual bool [is\\_equal](#) (const int &val) const  
*Check whether or not the computed expression is equal to another value.*
- virtual bool [is\\_equal](#) (const double &val) const  
*Check whether or not the computed expression is equal to another value.*
- virtual bool [is\\_equal](#) (const [Error](#) &val) const  
*Check whether or not the computed expression is equal to another value.*
- virtual [GarbageCollected](#) [\\_\\_add](#) (const [GarbageCollected](#) &rhs) const  
*Compute the result of adding this value and the supplied value.*
- virtual [GarbageCollected](#) [\\_\\_subtract](#) (const [GarbageCollected](#) &rhs) const  
*Compute the result of subtracting this value and the supplied value.*
- virtual [GarbageCollected](#) [\\_\\_multiply](#) (const [GarbageCollected](#) &rhs) const  
*Compute the result of multiplying this value and the supplied value.*
- virtual [GarbageCollected](#) [\\_\\_divide](#) (const [GarbageCollected](#) &rhs) const  
*Compute the result of dividing this value and the supplied value.*
- virtual [GarbageCollected](#) [\\_\\_modulo](#) (const [GarbageCollected](#) &rhs) const  
*Compute the result of moduloing this value and the supplied value.*
- virtual [GarbageCollected](#) [\\_\\_negative](#) () const  
*Compute the result of negating this value.*
- virtual [GarbageCollected](#) [\\_\\_lessThan](#) (const [GarbageCollected](#) &rhs) const  
*Compute the "less than" comparison.*

### 5.10.1 Detailed Description

Represents an Boolean that is the result of a computation.

### 5.10.2 Constructor & Destructor Documentation



### 5.10.2.1 ComputedExpressionBoolean()

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

Construct an Boolean result.

## Parameters

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

### 5.10.3 Member Function Documentation

#### 5.10.3.1 `__add()`

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

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

## Parameters

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

## Returns

The result of the operation.

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

#### 5.10.3.2 `__boolean()`

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

Perform a type cast to boolean.

## Returns

The result of the the operation.

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

#### 5.10.3.3 `__divide()`

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

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

## Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to divide this by.
------------	---

## Returns

The result of the operation.

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

### 5.10.3.4 `__equal()`

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

Perform an equalit test.

## Parameters

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

## Returns

The result of the the operation.

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

### 5.10.3.5 `__float()`

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

Perform a type cast to float.

## Returns

The result of the the operation.

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

### 5.10.3.6 `__integer()`

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

Perform a type cast to integer.

#### Returns

The result of the the operation.

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

### 5.10.3.7 `__lessThan()`

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

Compute the "less than" comparison.

#### Parameters

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

#### Returns

The result of the the operation.

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

### 5.10.3.8 `__modulo()`

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

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

#### Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to modulo this by.
------------	---

#### Returns

The result of the operation.

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

### 5.10.3.9 \_\_multiply()

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

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

#### Parameters

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

#### Returns

The result of the operation.

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

### 5.10.3.10 \_\_negative()

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

Compute the result of negating this value.

#### Returns

The result of the operation.

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

### 5.10.3.11 \_\_not()

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

Compute the logical not of this value.

#### Returns

The result of the operation.

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

### 5.10.3.12 \_\_subtract()

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

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

## Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to subtract from this.
------------	---

## Returns

The result of the operation.

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

**5.10.3.13 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](#).

**5.10.3.14 is\_equal() [1/4]**

```
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](#).

**5.10.3.15 is\_equal() [2/4]**

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

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

**Parameters**

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

**Returns**

True if equal, false if not.

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

**5.10.3.16 is\_equal() [3/4]**

```
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](#).

**5.10.3.17 is\_equal() [4/4]**

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

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

**Parameters**

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

**Returns**

True if equal, false if not.

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

### 5.10.3.18 makeCopy()

```
ComputedExpression * ComputedExpressionBoolean::makeCopy ( ) const [override], [virtual]
```

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

#### Returns

A pointer to the new [ComputedExpression](#).

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

Here is the call graph for this function:



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

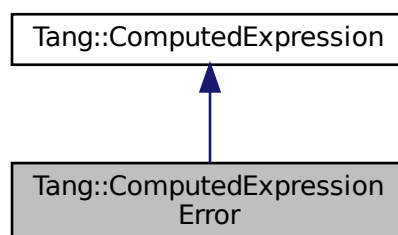
- [include/computedExpressionBoolean.hpp](#)
- [src/computedExpressionBoolean.cpp](#)

## 5.11 Tang::ComputedExpressionError Class Reference

Represents a Runtime [Error](#).

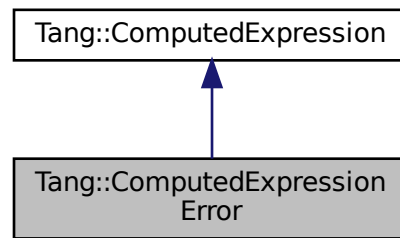
```
#include <computedExpressionError.hpp>
```

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.
- `ComputedExpression * makeCopy` () const override  
Make a copy of the *ComputedExpression* (recursively, if appropriate).
- virtual `bool is_equal` (const `Error` &val) const override  
Check whether or not the computed expression is equal to another value.
- virtual `GarbageCollected __add` (const `GarbageCollected` &rhs) const override  
Compute the result of adding this value and the supplied value.
- virtual `GarbageCollected __subtract` (const `GarbageCollected` &rhs) const override  
Compute the result of subtracting this value and the supplied value.
- virtual `GarbageCollected __multiply` (const `GarbageCollected` &rhs) const override  
Compute the result of multiplying this value and the supplied value.
- virtual `GarbageCollected __divide` (const `GarbageCollected` &rhs) const override  
Compute the result of dividing this value and the supplied value.
- virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const override  
Compute the result of moduloing this value and the supplied value.
- virtual `GarbageCollected __negative` () const override  
Compute the result of negating this value.
- virtual `GarbageCollected __not` () const override  
Compute the logical not of this value.
- virtual `GarbageCollected __lessThan` (const `GarbageCollected` &rhs) const override  
Compute the "less than" comparison.
- virtual `GarbageCollected __equal` (const `GarbageCollected` &rhs) const override  
Perform an equalit test.
- virtual `GarbageCollected __integer` () const override  
Perform a type cast to integer.
- virtual `GarbageCollected __float` () const override  
Perform a type cast to float.
- virtual `GarbageCollected __boolean` () const override  
Perform a type cast to boolean.

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

### 5.11.1 Detailed Description

Represents a Runtime [Error](#).

### 5.11.2 Constructor & Destructor Documentation

#### 5.11.2.1 ComputedExpressionError()

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

Construct a Runtime [Error](#).

##### Parameters

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

### 5.11.3 Member Function Documentation

#### 5.11.3.1 \_\_add()

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

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

##### Parameters

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

##### Returns

The result of the operation.

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

### 5.11.3.2 \_\_boolean()

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

Perform a type cast to boolean.

#### Returns

The result of the the operation.

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

### 5.11.3.3 \_\_divide()

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

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

#### Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to divide this by.
------------	---

#### Returns

The result of the operation.

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

### 5.11.3.4 \_\_equal()

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

Perform an equalit test.

#### Parameters

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

**Returns**

The result of the the operation.

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

**5.11.3.5 \_\_float()**

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

Perform a type cast to float.

**Returns**

The result of the the operation.

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

**5.11.3.6 \_\_integer()**

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

Perform a type cast to integer.

**Returns**

The result of the the operation.

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

**5.11.3.7 \_\_lessThan()**

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

Compute the "less than" comparison.

**Parameters**

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

### Returns

The result of the the operation.

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

#### 5.11.3.8 \_\_modulo()

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

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

### Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to modulo this by.
------------	---

### Returns

The result of the operation.

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

#### 5.11.3.9 \_\_multiply()

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

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

### Parameters

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

### Returns

The result of the operation.

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

#### 5.11.3.10 \_\_negative()

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

Compute the result of negating this value.

**Returns**

The result of the operation.

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

**5.11.3.11 \_\_not()**

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

Compute the logical not of this value.

**Returns**

The result of the operation.

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

**5.11.3.12 \_\_subtract()**

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

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

**Parameters**

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to subtract from this.
------------	---

**Returns**

The result of the operation.

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

**5.11.3.13 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](#).

#### 5.11.3.14 is\_equal() [1/4]

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

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

##### Parameters

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

##### Returns

True if equal, false if not.

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

#### 5.11.3.15 is\_equal() [2/4]

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

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

##### Parameters

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

##### Returns

True if equal, false if not.

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

#### 5.11.3.16 is\_equal() [3/4]

```
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](#).

**5.11.3.17 is\_equal()** [4/4]

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

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

**Parameters**

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

**Returns**

True if equal, false if not.

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

**5.11.3.18 makeCopy()**

```
ComputedExpression * ComputedExpressionError::makeCopy ( ) const [override], [virtual]
```

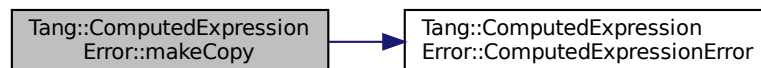
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

**Returns**

A pointer to the new [ComputedExpression](#).

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

Here is the call graph for this function:



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

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

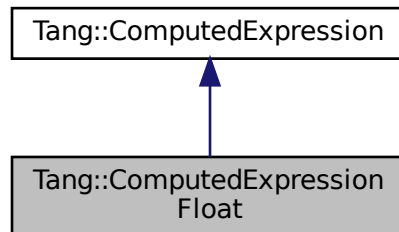


## 5.12 Tang::ComputedExpressionFloat Class Reference

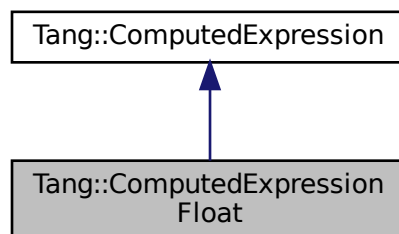
Represents a Float that is the result of a computation.

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

Inheritance diagram for Tang::ComputedExpressionFloat:



Collaboration diagram for Tang::ComputedExpressionFloat:



### Public Member Functions

- [ComputedExpressionFloat](#) (double val)  
*Construct a Float result.*
- virtual std::string [dump](#) () const override  
*Output the contents of the [ComputedExpression](#) as a string.*
- [ComputedExpression](#) \* [makeCopy](#) () const override  
*Make a copy of the [ComputedExpression](#) (recursively, if appropriate).*
- virtual bool [is\\_equal](#) (const int &val) const override  
*Check whether or not the computed expression is equal to another value.*
- virtual bool [is\\_equal](#) (const double &val) const override  
*Check whether or not the computed expression is equal to another value.*

- virtual `GarbageCollected __add` (const `GarbageCollected` &rhs) const override  
*Compute the result of adding this value and the supplied value.*
- virtual `GarbageCollected __subtract` (const `GarbageCollected` &rhs) const override  
*Compute the result of subtracting this value and the supplied value.*
- virtual `GarbageCollected __multiply` (const `GarbageCollected` &rhs) const override  
*Compute the result of multiplying this value and the supplied value.*
- virtual `GarbageCollected __divide` (const `GarbageCollected` &rhs) const override  
*Compute the result of dividing this value and the supplied value.*
- virtual `GarbageCollected __negative` () const override  
*Compute the result of negating this value.*
- virtual `GarbageCollected __not` () const override  
*Compute the logical not of this value.*
- virtual `GarbageCollected __lessThan` (const `GarbageCollected` &rhs) const override  
*Compute the "less than" comparison.*
- virtual `GarbageCollected __equal` (const `GarbageCollected` &rhs) const override  
*Perform an equalit test.*
- virtual `GarbageCollected __integer` () const override  
*Perform a type cast to integer.*
- virtual `GarbageCollected __float` () const override  
*Perform a type cast to float.*
- virtual `GarbageCollected __boolean` () const override  
*Perform a type cast to boolean.*
- virtual bool `is_equal` (const bool &val) const  
*Check whether or not the computed expression is equal to another value.*
- virtual bool `is_equal` (const `Error` &val) const  
*Check whether or not the computed expression is equal to another value.*
- virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const  
*Compute the result of moduloing this value and the supplied value.*

## Friends

- class `ComputedExpressionInteger`

### 5.12.1 Detailed Description

Represents a Float that is the result of a computation.

### 5.12.2 Constructor & Destructor Documentation

#### 5.12.2.1 ComputedExpressionFloat()

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

Construct a Float result.

## Parameters

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

### 5.12.3 Member Function Documentation

#### 5.12.3.1 \_\_add()

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

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

## Parameters

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

## Returns

The result of the operation.

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

#### 5.12.3.2 \_\_boolean()

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

Perform a type cast to boolean.

## Returns

The result of the the operation.

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

#### 5.12.3.3 \_\_divide()

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

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

**Parameters**

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to divide this by.
------------	---

**Returns**

The result of the operation.

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

**5.12.3.4 `__equal()`**

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

Perform an equalit test.

**Parameters**

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

**Returns**

The result of the the operation.

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

**5.12.3.5 `__float()`**

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

Perform a type cast to float.

**Returns**

The result of the the operation.

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

### 5.12.3.6 \_\_integer()

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

Perform a type cast to integer.

#### Returns

The result of the the operation.

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

### 5.12.3.7 \_\_lessThan()

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

Compute the "less than" comparison.

#### Parameters

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

#### Returns

The result of the the operation.

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

### 5.12.3.8 \_\_modulo()

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

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

#### Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to modulo this by.
------------	---

#### Returns

The result of the operation.

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

### 5.12.3.9 `__multiply()`

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

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

#### Parameters

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

#### Returns

The result of the operation.

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

### 5.12.3.10 `__negative()`

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

Compute the result of negating this value.

#### Returns

The result of the operation.

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

### 5.12.3.11 `__not()`

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

Compute the logical not of this value.

#### Returns

The result of the operation.

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

### 5.12.3.12 `__subtract()`

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

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

## Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to subtract from this.
------------	---

## Returns

The result of the operation.

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

**5.12.3.13 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](#).

**5.12.3.14 is\_equal() [1/4]**

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

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

## Parameters

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

## Returns

True if equal, false if not.

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

**5.12.3.15 is\_equal() [2/4]**

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

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

**Parameters**

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

**Returns**

True if equal, false if not.

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

**5.12.3.16 is\_equal() [3/4]**

```
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](#).

**5.12.3.17 is\_equal() [4/4]**

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

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

**Parameters**

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

**Returns**

True if equal, false if not.

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



### 5.12.3.18 makeCopy()

```
ComputedExpression * ComputedExpressionFloat::makeCopy ( ) const [override], [virtual]
```

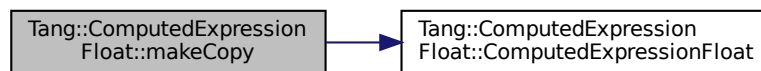
Make a copy of the [ComputedExpression](#) (recursively, if appropriate).

#### Returns

A pointer to the new [ComputedExpression](#).

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

Here is the call graph for this function:



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

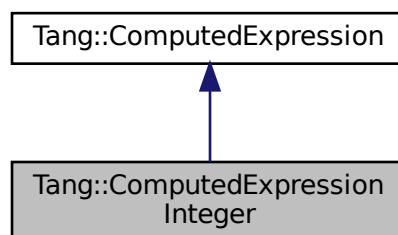
- [include/computedExpressionFloat.hpp](#)
- [src/computedExpressionFloat.cpp](#)

## 5.13 Tang::ComputedExpressionInteger Class Reference

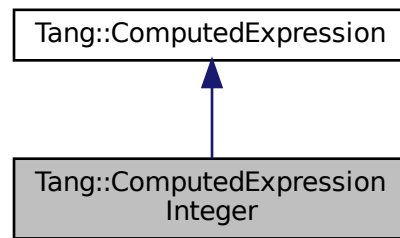
Represents an Integer that is the result of a computation.

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

Inheritance diagram for `Tang::ComputedExpressionInteger`:



Collaboration diagram for Tang::ComputedExpressionInteger:



## Public Member Functions

- `ComputedExpressionInteger` (int64\_t val)  
*Construct an Integer result.*
- virtual `std::string dump` () const override  
*Output the contents of the `ComputedExpression` as a string.*
- `ComputedExpression * makeCopy` () const override  
*Make a copy of the `ComputedExpression` (recursively, if appropriate).*
- virtual `bool is_equal` (const int &val) const override  
*Check whether or not the computed expression is equal to another value.*
- virtual `bool is_equal` (const double &val) const override  
*Check whether or not the computed expression is equal to another value.*
- virtual `GarbageCollected __add` (const `GarbageCollected` &rhs) const override  
*Compute the result of adding this value and the supplied value.*
- virtual `GarbageCollected __subtract` (const `GarbageCollected` &rhs) const override  
*Compute the result of subtracting this value and the supplied value.*
- virtual `GarbageCollected __multiply` (const `GarbageCollected` &rhs) const override  
*Compute the result of multiplying this value and the supplied value.*
- virtual `GarbageCollected __divide` (const `GarbageCollected` &rhs) const override  
*Compute the result of dividing this value and the supplied value.*
- virtual `GarbageCollected __modulo` (const `GarbageCollected` &rhs) const override  
*Compute the result of moduloing this value and the supplied value.*
- virtual `GarbageCollected __negative` () const override  
*Compute the result of negating this value.*
- virtual `GarbageCollected __not` () const override  
*Compute the logical not of this value.*
- virtual `GarbageCollected __lessThan` (const `GarbageCollected` &rhs) const override  
*Compute the "less than" comparison.*
- virtual `GarbageCollected __equal` (const `GarbageCollected` &rhs) const override  
*Perform an equalit test.*
- virtual `GarbageCollected __integer` () const override  
*Perform a type cast to integer.*
- virtual `GarbageCollected __float` () const override  
*Perform a type cast to float.*

- virtual [GarbageCollected](#) [\\_\\_boolean](#) () const override  
*Perform a type cast to boolean.*
- virtual bool [is\\_equal](#) (const bool &val) const  
*Check whether or not the computed expression is equal to another value.*
- virtual bool [is\\_equal](#) (const [Error](#) &val) const  
*Check whether or not the computed expression is equal to another value.*

## Friends

- class [ComputedExpressionFloat](#)

### 5.13.1 Detailed Description

Represents an Integer that is the result of a computation.

### 5.13.2 Constructor & Destructor Documentation

#### 5.13.2.1 ComputedExpressionInteger()

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

Construct an Integer result.

##### Parameters

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

### 5.13.3 Member Function Documentation

#### 5.13.3.1 \_\_add()

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

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

##### Parameters

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

**Returns**

The result of the operation.

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

**5.13.3.2 \_\_boolean()**

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

Perform a type cast to boolean.

**Returns**

The result of the the operation.

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

**5.13.3.3 \_\_divide()**

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

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

**Parameters**

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to divide this by.
------------	---

**Returns**

The result of the operation.

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

**5.13.3.4 \_\_equal()**

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

Perform an equalit test.

## Parameters

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

## Returns

The result of the the operation.

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

**5.13.3.5 \_\_float()**

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

Perform a type cast to float.

## Returns

The result of the the operation.

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

**5.13.3.6 \_\_integer()**

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

Perform a type cast to integer.

## Returns

The result of the the operation.

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

**5.13.3.7 \_\_lessThan()**

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

Compute the "less than" comparison.

## Parameters

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

## Returns

The result of the the operation.

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

**5.13.3.8 \_\_modulo()**

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

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

## Parameters

<i>rhs</i>	The <a href="#">GarbageCollected</a> value to modulo this by.
------------	---

## Returns

The result of the operation.

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

**5.13.3.9 \_\_multiply()**

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

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

## Parameters

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

## Returns

The result of the operation.

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

#### 5.13.3.10 \_\_negative()

`GarbageCollected` ComputedExpressionInteger::\_\_negative ( ) const [override], [virtual]

Compute the result of negating this value.

##### Returns

The result of the operation.

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

#### 5.13.3.11 \_\_not()

`GarbageCollected` ComputedExpressionInteger::\_\_not ( ) const [override], [virtual]

Compute the logical not of this value.

##### Returns

The result of the operation.

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

#### 5.13.3.12 \_\_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 <code>GarbageCollected</code> value to subtract from this.
------------	--

##### Returns

The result of the operation.

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

#### 5.13.3.13 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](#).

#### 5.13.3.14 is\_equal() [1/4]

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

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

##### Parameters

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

##### Returns

True if equal, false if not.

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

#### 5.13.3.15 is\_equal() [2/4]

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

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

##### Parameters

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

##### Returns

True if equal, false if not.

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



#### 5.13.3.16 is\_equal() [3/4]

```
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](#).

#### 5.13.3.17 is\_equal() [4/4]

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

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

##### Parameters

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

##### Returns

True if equal, false if not.

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

#### 5.13.3.18 makeCopy()

```
ComputedExpression * ComputedExpressionInteger::makeCopy ( ) const [override], [virtual]
```

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

**Returns**

A pointer to the new [ComputedExpression](#).

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

Here is the call graph for this function:



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

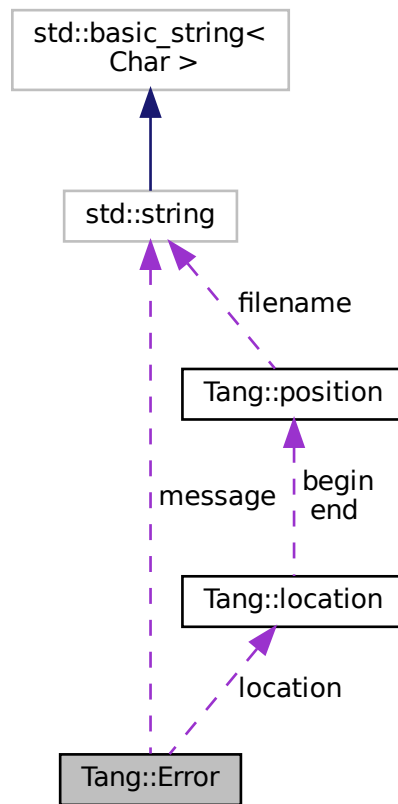
- [include/computedExpressionInteger.hpp](#)
- [src/computedExpressionInteger.cpp](#)

## 5.14 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.*

### 5.14.1 Detailed Description

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

### 5.14.2 Constructor & Destructor Documentation

#### 5.14.2.1 [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.
----------------	--------------------------------

#### 5.14.2.2 [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.

### 5.14.3 Friends And Related Function Documentation

### 5.14.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & out,
    const Error & error ) [friend]
```

Add friendly output.

#### Parameters

<i>out</i>	The output stream.
<i>error</i>	The <a href="#">Error</a> object.

#### Returns

The output stream.

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

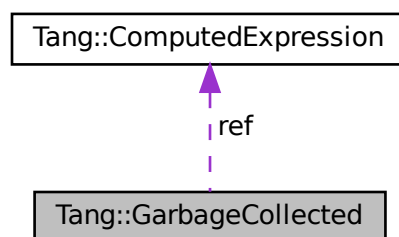
- [include/error.hpp](#)
- [src/error.cpp](#)

## 5.15 Tang::GarbageCollected Class Reference

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

```
#include <garbageCollected.hpp>
```

Collaboration diagram for Tang::GarbageCollected:



## Public Member Functions

- [GarbageCollected](#) (const [GarbageCollected](#) &other)  
*Copy Constructor.*
- [GarbageCollected](#) ([GarbageCollected](#) &&other)  
*Move Constructor.*
- [GarbageCollected](#) & operator= (const [GarbageCollected](#) &other)  
*Copy Assignment.*
- [GarbageCollected](#) & operator= ([GarbageCollected](#) &&other)  
*Move Assignment.*
- [~GarbageCollected](#) ()  
*Destructor.*
- [ComputedExpression](#) \* operator-> () const  
*Access the tracked object as a pointer.*
- [ComputedExpression](#) & operator\* () const  
*Access the tracked object.*
- bool operator== (const int &val) const  
*Compare the [GarbageCollected](#) tracked object with a supplied value.*
- bool operator== (const double &val) const  
*Compare the [GarbageCollected](#) tracked object with a supplied value.*
- bool operator== (const bool &val) const  
*Compare the [GarbageCollected](#) tracked object with a supplied value.*
- bool operator== (const [Error](#) &val) 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.*

### 5.15.1 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.

### 5.15.2 Constructor & Destructor Documentation

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

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

Copy Constructor.

Parameters

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

### 5.15.2.2 `GarbageCollected()` [2/3]

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

Move Constructor.

#### Parameters

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

### 5.15.2.3 `~GarbageCollected()`

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

Destructor.

Clean up the tracked object, if appropriate.

### 5.15.2.4 `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.
-----------------	---

## 5.15.3 Member Function Documentation

### 5.15.3.1 `make()`

```
template<class T , typename... Args>
static GarbageCollected Tang::GarbageCollected::make (
    Args... args ) [inline], [static]
```

Creates a garbage-collected object of the specified type.



## Parameters

<i>variable</i>	The arguments to pass to the constructor of the specified type.
-----------------	---

## Returns

A [GarbageCollected](#) object.

Here is the call graph for this function:



## 5.15.3.2 operator"!")()

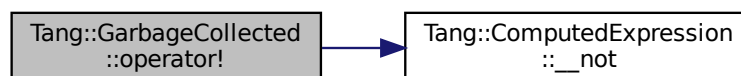
```
GarbageCollected GarbageCollected::operator! ( ) const
```

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

## Returns

The result of the operation.

Here is the call graph for this function:



## 5.15.3.3 operator"!=( )

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

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

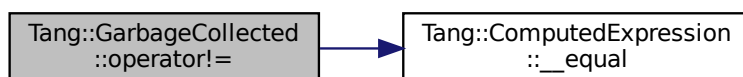
**Parameters**

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

**Returns**

The result of the operation.

Here is the call graph for this function:

**5.15.3.4 operator%()**

```

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

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

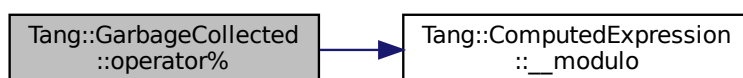
**Parameters**

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

**Returns**

The result of the operation.

Here is the call graph for this function:



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

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

Access the tracked object.

**Returns**

A reference to the tracked object.

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

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

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

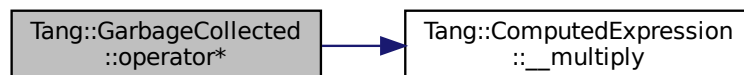
**Parameters**

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

**Returns**

The result of the operation.

Here is the call graph for this function:

**5.15.3.7 operator+()**

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

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

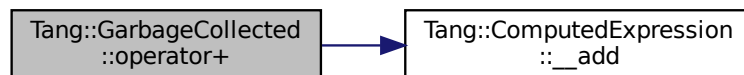
## Parameters

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

## Returns

The result of the operation.

Here is the call graph for this function:

5.15.3.8 **operator-()** [1/2]

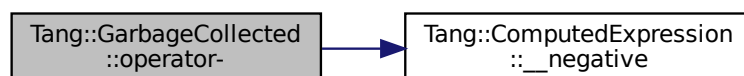
```
GarbageCollected GarbageCollected::operator- ( ) const
```

Perform a negation on the [GarbageCollected](#) value.

## Returns

The result of the operation.

Here is the call graph for this function:

5.15.3.9 **operator-()** [2/2]

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

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

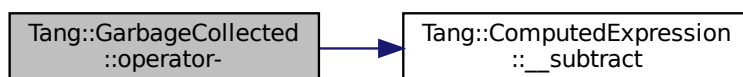
## Parameters

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

## Returns

The result of the operation.

Here is the call graph for this function:



## 5.15.3.10 operator-&gt;()

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

Access the tracked object as a pointer.

## Returns

A pointer to the tracked object.

## 5.15.3.11 operator/()

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

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

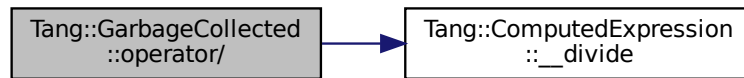
## Parameters

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

## Returns

The result of the operation.

Here is the call graph for this function:



#### 5.15.3.12 operator<()

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

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

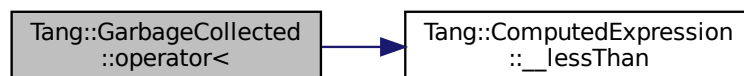
##### Parameters

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

##### Returns

The result of the operation.

Here is the call graph for this function:



#### 5.15.3.13 operator<=()

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

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

## Parameters

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

## Returns

The result of the operation.

## 5.15.3.14 operator=() [1/2]

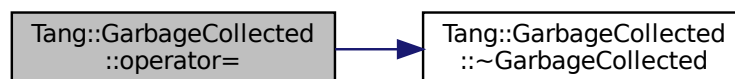
```
GarbageCollected& Tang::GarbageCollected::operator= (
    const GarbageCollected & other ) [inline]
```

Copy Assignment.

## Parameters

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

Here is the call graph for this function:



## 5.15.3.15 operator=() [2/2]

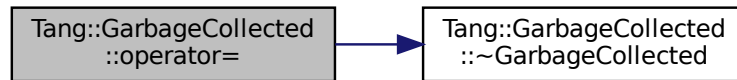
```
GarbageCollected& Tang::GarbageCollected::operator= (
    GarbageCollected && other ) [inline]
```

Move Assignment.

## Parameters

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

Here is the call graph for this function:



#### 5.15.3.16 `operator==( )` [1/5]

```
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.

#### 5.15.3.17 `operator==( )` [2/5]

```
bool GarbageCollected::operator== (
    const double & val ) const
```

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

##### Parameters

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

##### Returns

True if they are equal, false otherwise.



**5.15.3.18 operator==( ) [3/5]**

```
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.

**5.15.3.19 operator==( ) [4/5]**

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

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

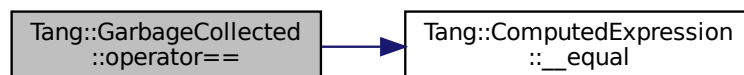
**Parameters**

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

**Returns**

The result of the operation.

Here is the call graph for this function:

**5.15.3.20 operator==( ) [5/5]**

```
bool GarbageCollected::operator==(
    const int & val ) const
```

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

**Parameters**

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

**Returns**

True if they are equal, false otherwise.

**5.15.3.21 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.

**5.15.3.22 operator>=()**

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

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

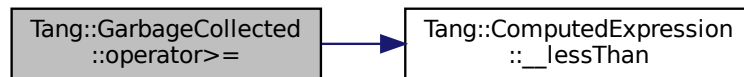
**Parameters**

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

**Returns**

The result of the operation.

Here is the call graph for this function:



## 5.15.4 Friends And Related Function Documentation

### 5.15.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & out,
    const GarbageCollected & gc ) [friend]
```

Add friendly output.

**Parameters**

<i>out</i>	The output stream.
<i>gc</i>	The <a href="#">GarbageCollected</a> value.

**Returns**

The output stream.

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

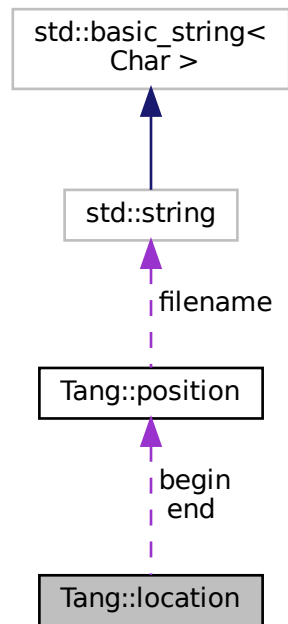
- [include/garbageCollected.hpp](#)
- [src/garbageCollected.cpp](#)

## 5.16 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.*

### 5.16.1 Detailed Description

Two points in a source file.

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

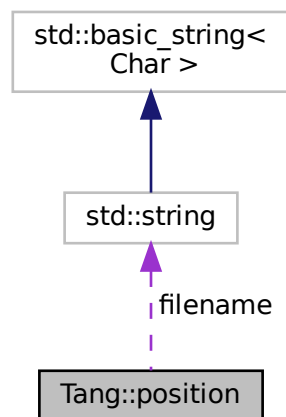
- build/generated/[location.hh](#)

## 5.17 Tang::position Class Reference

A point in a source file.

```
#include <location.hh>
```

Collaboration diagram for Tang::position:



## Public Types

- typedef const std::string [filename\\_type](#)  
*Type for file name.*
- typedef int [counter\\_type](#)  
*Type for line and column numbers.*

## Public Member Functions

- [position](#) ([filename\\_type](#) \*f=((void \*) 0), [counter\\_type](#) l=1, [counter\\_type](#) c=1)  
*Construct a position.*
- void [initialize](#) ([filename\\_type](#) \*fn=((void \*) 0), [counter\\_type](#) l=1, [counter\\_type](#) c=1)  
*Initialization.*

### Line and Column related manipulators

- void [lines](#) ([counter\\_type](#) count=1)  
*(line related) Advance to the COUNT next lines.*
- void [columns](#) ([counter\\_type](#) count=1)  
*(column related) Advance to the COUNT next columns.*

## Public Attributes

- [filename\\_type](#) \* [filename](#)  
*File name to which this position refers.*
- [counter\\_type](#) [line](#)  
*Current line number.*
- [counter\\_type](#) [column](#)  
*Current column number.*

### 5.17.1 Detailed Description

A point in a source file.

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

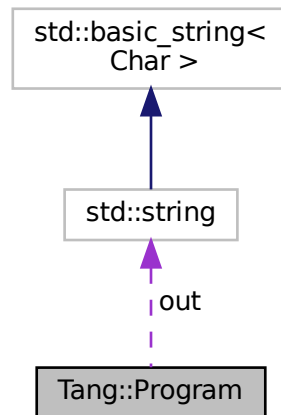
- build/generated/[location.hh](#)

## 5.18 Tang::Program Class Reference

Represents a compiled script or template that may be executed.

```
#include <program.hpp>
```

Collaboration diagram for Tang::Program:



## Public Types

- enum `CodeType` { `Script` , `Template` }  
Indicate the type of code that was supplied to the `Program`.

## Public Member Functions

- `Program` (`std::string` code, `CodeType` codeType)  
Create a compiled program using the provided code.
- `~Program` ()  
The `Program` Destructor.
- `Program` (const `Program` &program)  
The Copy Constructor.
- `Program` & `operator=` (const `Program` &program)  
The Copy Assignment operator.
- `Program` (`Program` &&program)  
The Move Constructor.
- `Program` & `operator=` (`Program` &&program)  
The Move Assignment operator.
- `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.
- void `addBytecode` (uint64\_t)  
Add a `uint64_t` to the Bytecode.
- `Program` & `execute` ()  
Execute the program's Bytecode, and return the current `Program` object.

## Public Attributes

- `std::string out`

*The output of the program, resulting from the program execution.*

### 5.18.1 Detailed Description

Represents a compiled script or template that may be executed.

### 5.18.2 Member Enumeration Documentation

#### 5.18.2.1 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.

### 5.18.3 Constructor & Destructor Documentation

#### 5.18.3.1 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> .

### 5.18.4 Member Function Documentation



#### 5.18.4.1 addBytecode()

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

Add a `uint64_t` to the Bytecode.

##### Parameters

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

#### 5.18.4.2 dumpBytecode()

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

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

##### Returns

A string containing the Opcode representation.

#### 5.18.4.3 execute()

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

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

##### Returns

The current [Program](#) object.

#### 5.18.4.4 getAst()

```
optional< const shared_ptr< AstNode > > Program::getAst ( ) const
```

Get the AST that was generated by the parser.

The parser may have failed, so the return is an `optional<>` type. If the compilation failed, check `Program::error`.

##### Returns

A pointer to the AST, if it exists.

#### 5.18.4.5 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.

#### 5.18.4.6 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.

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](#)

## 5.19 [Tang::SingletonObjectPool< T >](#) Class Template Reference

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

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

### Public Member Functions

- [T \\* get](#) ()  
*Request an uninitialized memory location from the pool for an object T.*
- void [recycle](#) (T \*obj)  
*Recycle a memory location for an object T.*
- [~SingletonObjectPool](#) ()  
*Destructor.*

### Static Public Member Functions

- static [SingletonObjectPool< T > & getInstance](#) ()  
*Get the singleton instance of the object pool.*

### 5.19.1 Detailed Description

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

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

### 5.19.2 Member Function Documentation

#### 5.19.2.1 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.

#### 5.19.2.2 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.

#### 5.19.2.3 recycle()

```
template<class T >
void Tang::SingletonObjectPool< T >::recycle (
    T * obj ) [inline]
```

Recycle a memory location for an object T.

## Parameters

<i>obj</i>	The memory location to recycle.
------------	---------------------------------

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

- include/[singletonObjectPool.hpp](#)

## 5.20 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](#).*

#### 5.20.1 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).

#### 5.20.2 Constructor & Destructor Documentation

##### 5.20.2.1 TangBase()

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

The constructor.

Isn't it glorious.

#### 5.20.3 Member Function Documentation

##### 5.20.3.1 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:

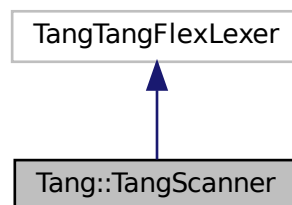
- [include/tangBase.hpp](#)
- [src/tangBase.cpp](#)

## 5.21 Tang::TangScanner Class Reference

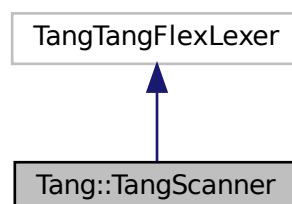
The Flex lexer class for the main Tang language.

```
#include <tangScanner.hpp>
```

Inheritance diagram for Tang::TangScanner:



Collaboration diagram for Tang::TangScanner:



## Public Member Functions

- [TangScanner](#) (std::istream &arg\_yyin, std::ostream &arg\_yyout)  
*The constructor for the Scanner.*
- virtual Tang::TangParser::symbol\_type [get\\_next\\_token](#) ()  
*A pass-through function that we supply so that we can provide a Bison 3 token return type instead of the `int` that is returned by the default class configuration.*

### 5.21.1 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.

### 5.21.2 Constructor & Destructor Documentation

#### 5.21.2.1 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)

### 5.21.3 Member Function Documentation

#### 5.21.3.1 get\_next\_token()

```
virtual Tang::TangParser::symbol_type Tang::TangScanner::get_next_token ( ) [virtual]
```

A pass-through function that we supply so that we can provide a Bison 3 token return type instead of the `int` that is returned by the default class configuration.

#### Returns

A Bison 3 token representing the lexeme that was recognized.

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

- include/[tangScanner.hpp](#)





## Chapter 6

# File Documentation

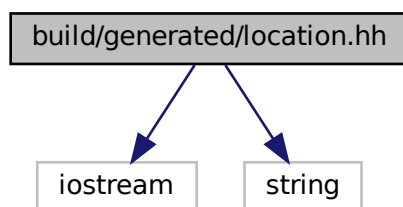
### 6.1 build/generated/location.hh File Reference

Define the Tang ::location class.

```
#include <iostream>
```

```
#include <string>
```

Include dependency graph for location.hh:



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



### Classes

- class [Tang::position](#)  
*A point in a source file.*
- class [Tang::location](#)  
*Two points in a source file.*

## Macros

- `#define YY_NULLPTR ((void*)0)`

## Functions

- position & [Tang::operator+=](#) (position &res, position::counter\_type width)  
*Add width columns, in place.*
- position [Tang::operator+](#) (position res, position::counter\_type width)  
*Add width columns.*
- position & [Tang::operator-=](#) (position &res, position::counter\_type width)  
*Subtract width columns, in place.*
- position [Tang::operator-](#) (position res, position::counter\_type width)  
*Subtract width columns.*
- template<typename YYChar >  
std::basic\_ostream< YYChar > & [Tang::operator<<](#) (std::basic\_ostream< YYChar > &ostr, const position &pos)  
*Intercept output stream redirection.*
- location & [Tang::operator+=](#) (location &res, const location &end)  
*Join two locations, in place.*
- location [Tang::operator+](#) (location res, const location &end)  
*Join two locations.*
- location & [Tang::operator+=](#) (location &res, location::counter\_type width)  
*Add width columns to the end position, in place.*
- location [Tang::operator+](#) (location res, location::counter\_type width)  
*Add width columns to the end position.*
- location & [Tang::operator-=](#) (location &res, location::counter\_type width)  
*Subtract width columns to the end position, in place.*
- location [Tang::operator-](#) (location res, location::counter\_type width)  
*Subtract width columns to the end position.*
- template<typename YYChar >  
std::basic\_ostream< YYChar > & [Tang::operator<<](#) (std::basic\_ostream< YYChar > &ostr, const location &loc)  
*Intercept output stream redirection.*

### 6.1.1 Detailed Description

Define the Tang ::location class.

### 6.1.2 Function Documentation

#### 6.1.2.1 operator<<() [1/2]

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

Intercept output stream redirection.

## Parameters

<i>ostr</i>	the destination output stream
<i>loc</i>	a reference to the location to redirect

Avoid duplicate information.

## 6.1.2.2 operator&lt;&lt;() [2/2]

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

Intercept output stream redirection.

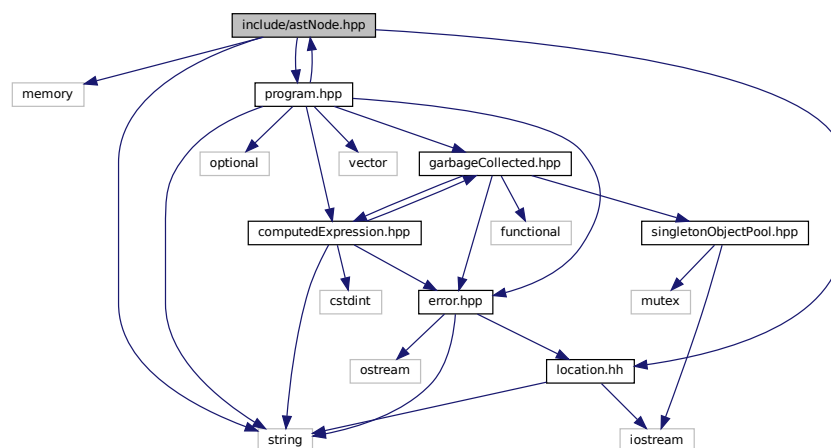
## Parameters

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

## 6.2 include/astNode.hpp File Reference

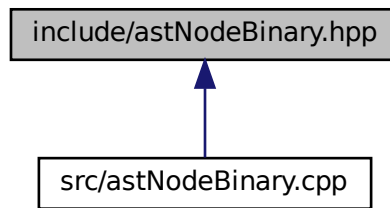
Declare the [Tang::AstNode](#) base class.

```
#include <memory>
#include <string>
#include "location.hh"
#include "program.hpp"
Include dependency graph for astNode.hpp:
```





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



## Classes

- class [Tang::AstNodeBinary](#)  
An [AstNode](#) that represents a binary expression.

### 6.3.1 Detailed Description

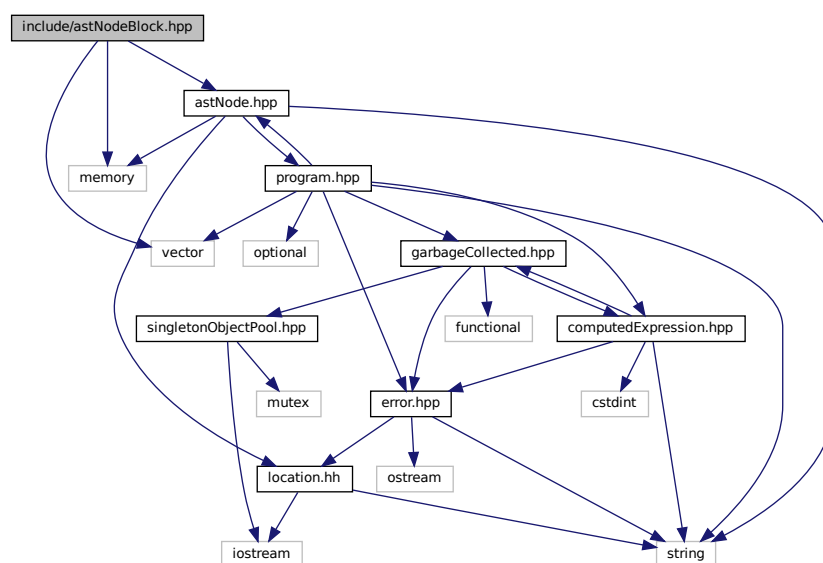
Declare the [Tang::AstNodeBinary](#) class.

## 6.4 include/astNodeBlock.hpp File Reference

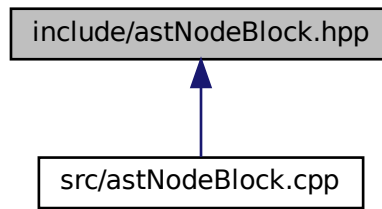
Declare the [Tang::AstNodeBlock](#) class.

```
#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.

### 6.4.1 Detailed Description

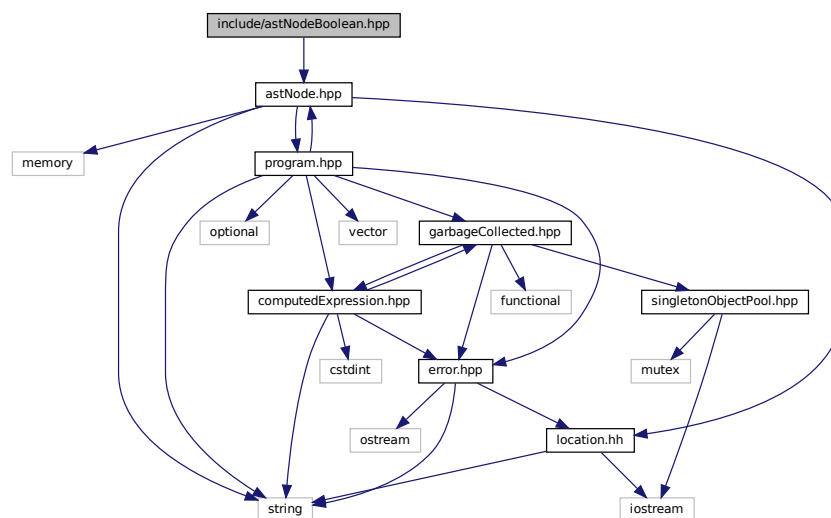
Declare the [Tang::AstNodeBlock](#) class.

## 6.5 include/astNodeBoolean.hpp File Reference

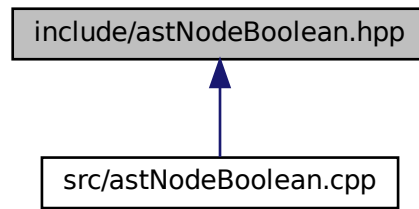
Declare the [Tang::AstNodeBoolean](#) class.

```
#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.*

### 6.5.1 Detailed Description

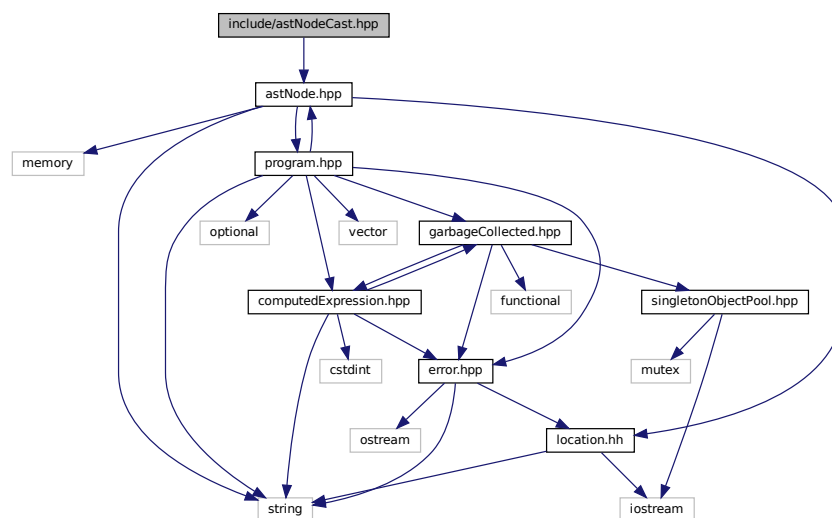
Declare the [Tang::AstNodeBoolean](#) class.

## 6.6 include/astNodeCast.hpp File Reference

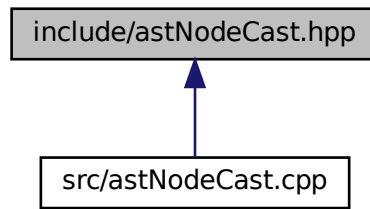
Declare the [Tang::AstNodeCast](#) class.

```
#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.

### 6.6.1 Detailed Description

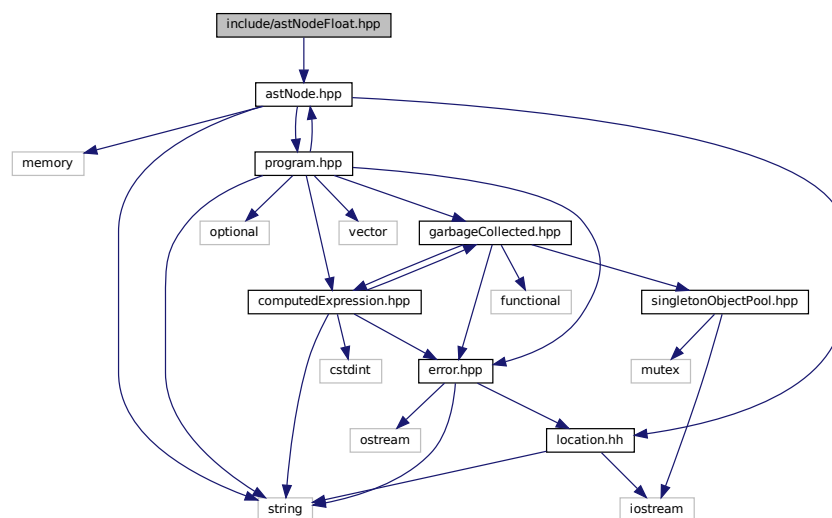
Declare the [Tang::AstNodeCast](#) class.

## 6.7 include/astNodeFloat.hpp File Reference

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

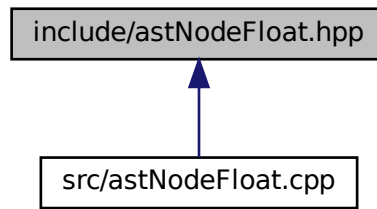
```
#include "astNode.hpp"
```

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.*

### 6.7.1 Detailed Description

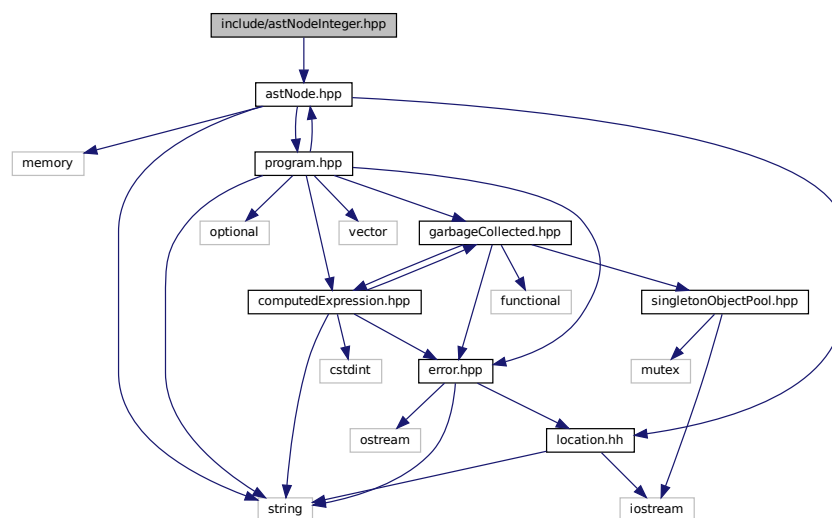
Declare the [Tang::AstNodeFloat](#) class.

## 6.8 include/astNodeInteger.hpp File Reference

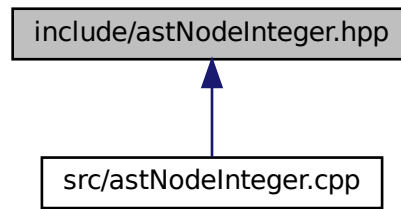
Declare the [Tang::AstNodeInteger](#) class.

```
#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.

### 6.8.1 Detailed Description

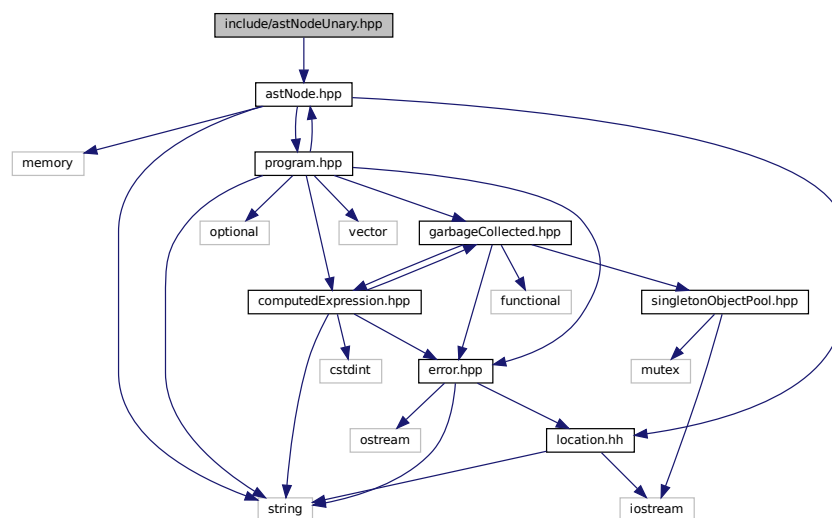
Declare the [Tang::AstNodeInteger](#) class.

## 6.9 include/astNodeUnary.hpp File Reference

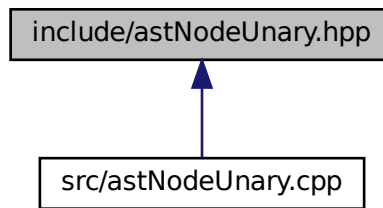
Declare the [Tang::AstNodeUnary](#) class.

```
#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.

### 6.9.1 Detailed Description

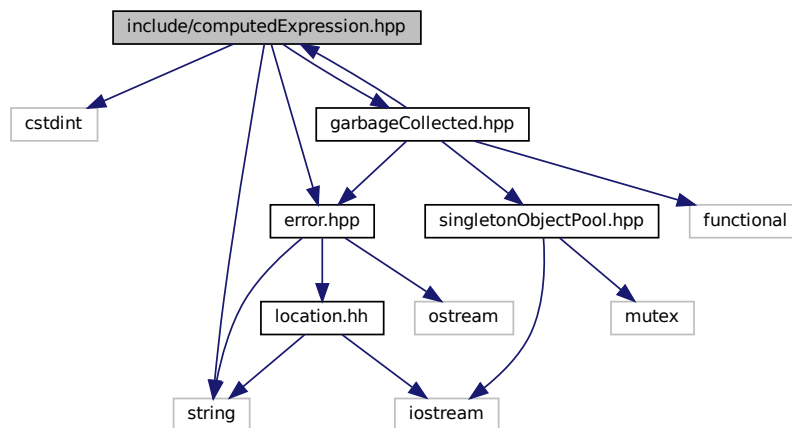
Declare the [Tang::AstNodeUnary](#) class.

## 6.10 include/computedExpression.hpp File Reference

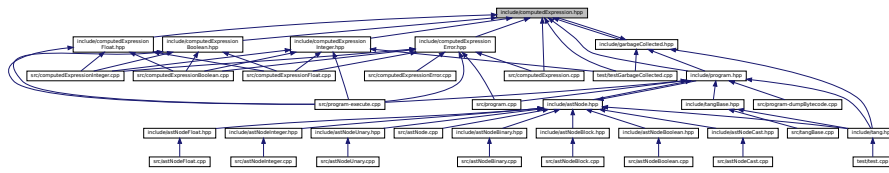
Declare the [Tang::ComputedExpression](#) base class.

```
#include <stdint>
#include <string>
#include "garbageCollected.hpp"
#include "error.hpp"
```

Include dependency graph for computedExpression.hpp:



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



## Classes

- class [Tang::ComputedExpression](#)

*Represents the result of a computation that has been executed.*

### 6.10.1 Detailed Description

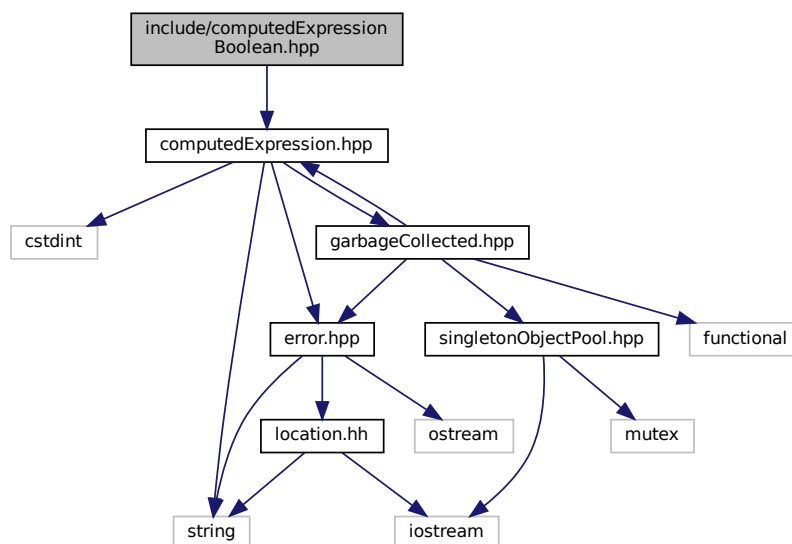
Declare the [Tang::ComputedExpression](#) base class.

## 6.11 include/computedExpressionBoolean.hpp File Reference

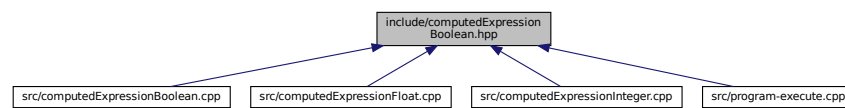
Declare the [Tang::ComputedExpressionBoolean](#) class.

```
#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.*

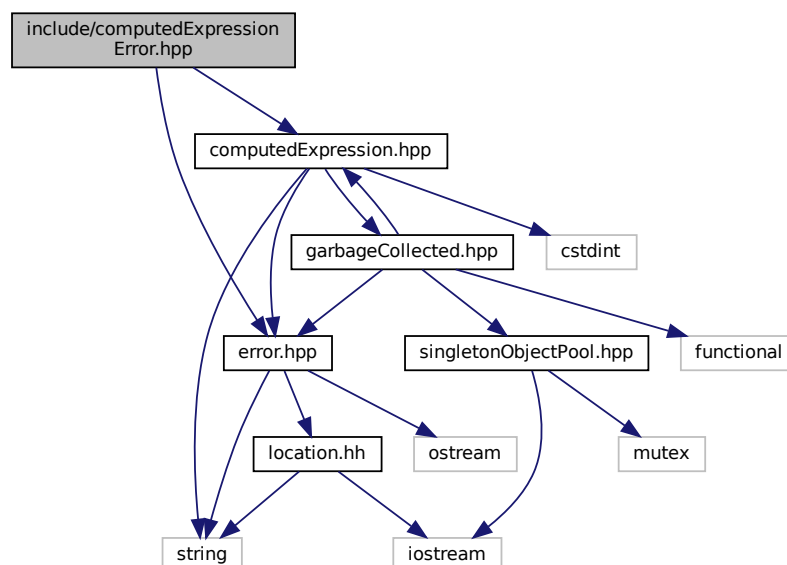
### 6.11.1 Detailed Description

Declare the `Tang::ComputedExpressionBoolean` class.

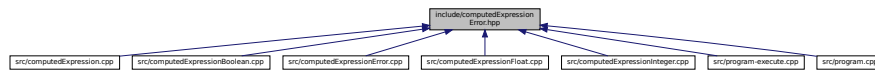
## 6.12 include/computedExpressionError.hpp File Reference

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](#).*

### 6.12.1 Detailed Description

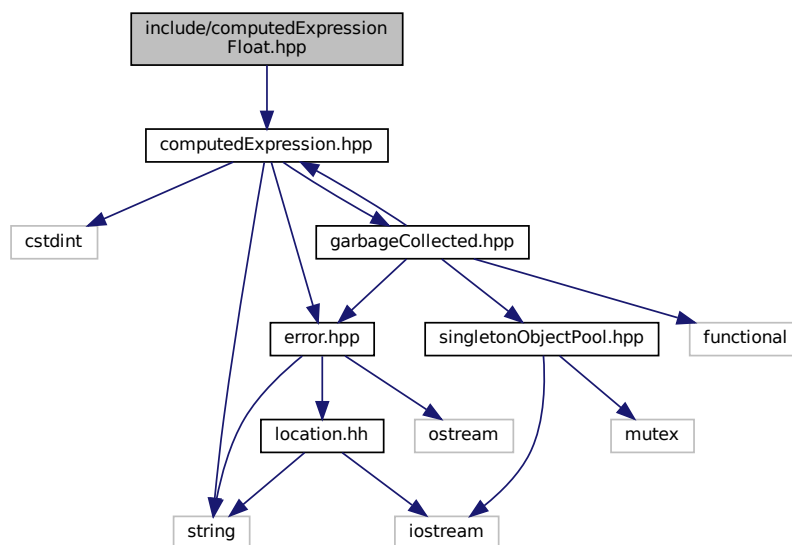
Declare the [Tang::ComputedExpressionError](#) class.

## 6.13 include/computedExpressionFloat.hpp File Reference

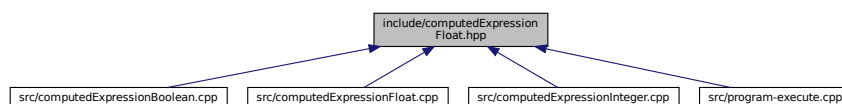
Declare the [Tang::ComputedExpressionFloat](#) class.

```
#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.*

### 6.13.1 Detailed Description

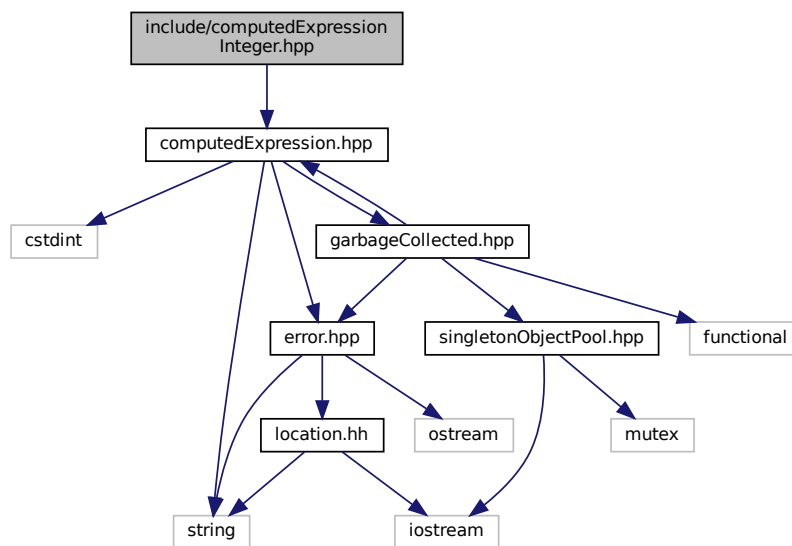
Declare the [Tang::ComputedExpressionFloat](#) class.

## 6.14 include/computedExpressionInteger.hpp File Reference

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

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

Include dependency graph for computedExpressionInteger.hpp:



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



## Classes

- class [Tang::ComputedExpressionInteger](#)  
*Represents an Integer that is the result of a computation.*





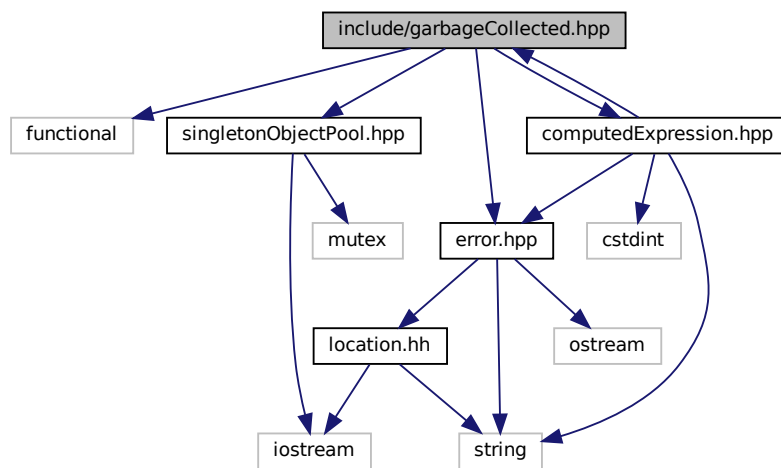
### 6.15.1 Detailed Description

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

## 6.16 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.*

### 6.16.1 Detailed Description

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

## 6.17 include/macros.hpp File Reference

Contains generic macros.

### Macros

- `#define TANG_UNUSED(x) x`

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

### 6.17.1 Detailed Description

Contains generic macros.

### 6.17.2 Macro Definition Documentation

#### 6.17.2.1 TANG\_UNUSED

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

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

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

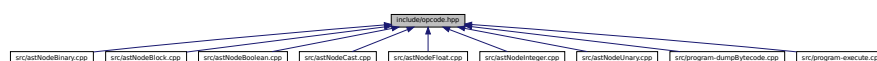
#### Parameters

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

## 6.18 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`, `INTEGER`, `FLOAT`, `BOOLEAN`,  
`ADD`, `SUBTRACT`, `MULTIPLY`, `DIVIDE`,  
`MODULO`, `NEGATIVE`, `NOT`, `LT`,  
`LTE`, `GT`, `GTE`, `EQ`,  
`NEQ`, `CASTINTEGER`, `CASTFLOAT`, `CASTBOOLEAN` }

### 6.18.1 Detailed Description

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

### 6.18.2 Enumeration Type Documentation

#### 6.18.2.1 Opcode

```
enum Tang::Opcode [strong]
```

Enumerator

<code>POP</code>	Pop a val.
<code>INTEGER</code>	Push an integer onto the stack.
<code>FLOAT</code>	Push a floating point number onto the stack.
<code>BOOLEAN</code>	Push a boolean onto the stack.
<code>ADD</code>	Pop rhs, pop lhs, push lhs + rhs.
<code>SUBTRACT</code>	Pop rhs, pop lhs, push lhs - rhs.
<code>MULTIPLY</code>	Pop rhs, pop lhs, push lhs * rhs.
<code>DIVIDE</code>	Pop rhs, pop lhs, push lhs / rhs.
<code>MODULO</code>	Pop rhs, pop lhs, push lhs % rhs.
<code>NEGATIVE</code>	Pop val, push negative val.
<code>NOT</code>	Pop val, push logical not of val.
<code>LT</code>	Pop rhs, pop lhs, push lhs < rhs.
<code>LTE</code>	Pop rhs, pop lhs, push lhs <= rhs.
<code>GT</code>	Pop rhs, pop lhs, push lhs > rhs.
<code>GTE</code>	Pop rhs, pop lhs, push lhs >= rhs.
<code>EQ</code>	Pop rhs, pop lhs, push lhs == rhs.
<code>NEQ</code>	Pop rhs, pop lhs, push lhs != rhs.
<code>CASTINTEGER</code>	Pop a val, typecast to int, push.
<code>CASTFLOAT</code>	Pop a val, typecast to float, push.
<code>CASTBOOLEAN</code>	Pop a val, typecast to boolean, push.

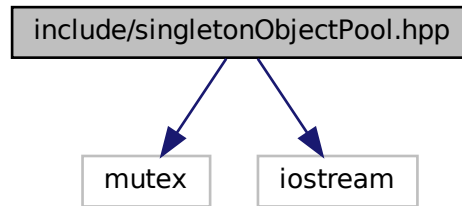


## 6.20 include/singletonObjectPool.hpp File Reference

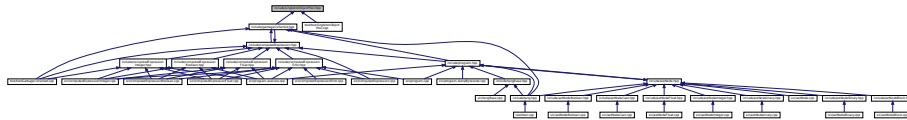
Declare the [Tang::SingletonObjectPool](#) class.

```
#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.

#### 6.20.1 Detailed Description

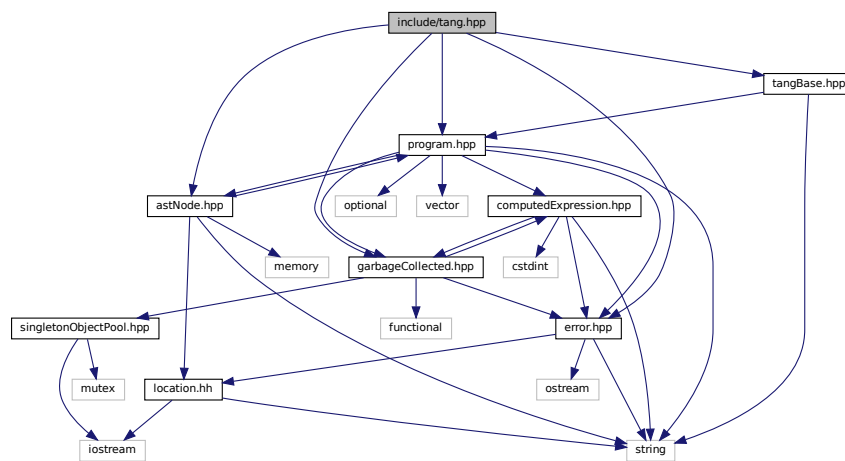
Declare the [Tang::SingletonObjectPool](#) class.

## 6.21 include/tang.hpp File Reference

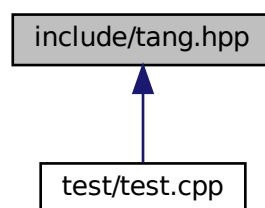
Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

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

Include dependency graph for tang.hpp:



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



### 6.21.1 Detailed Description

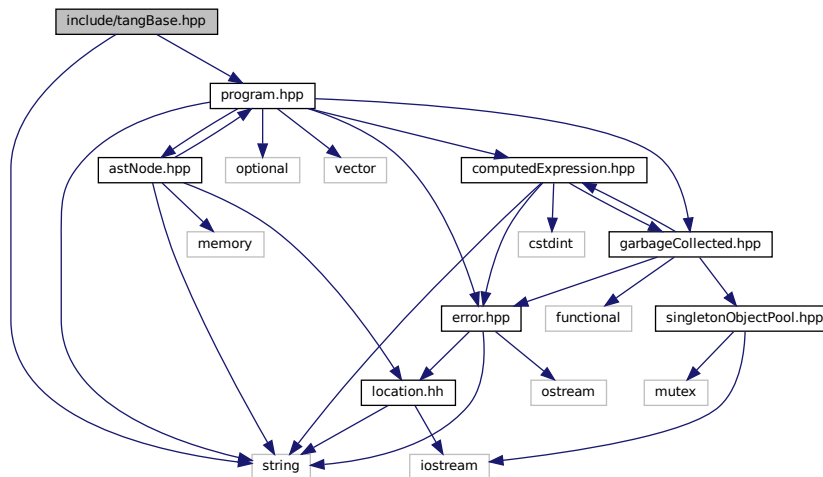
Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

## 6.22 include/tangBase.hpp File Reference

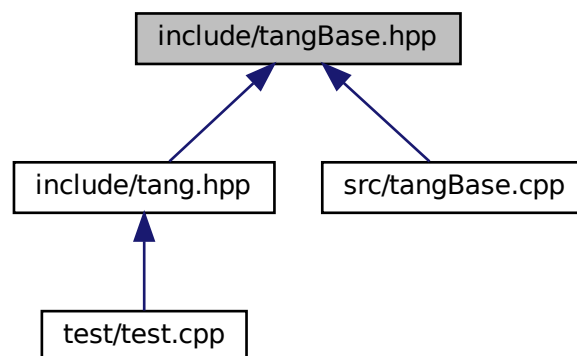
Declare the [Tang::TangBase](#) class used to interact with Tang.

```
#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.*

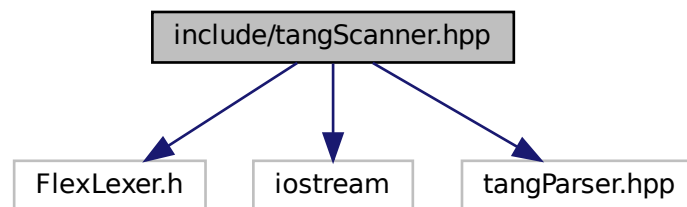
### 6.22.1 Detailed Description

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

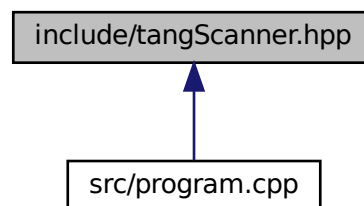
## 6.23 include/tangScanner.hpp File Reference

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()`

### 6.23.1 Detailed Description

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

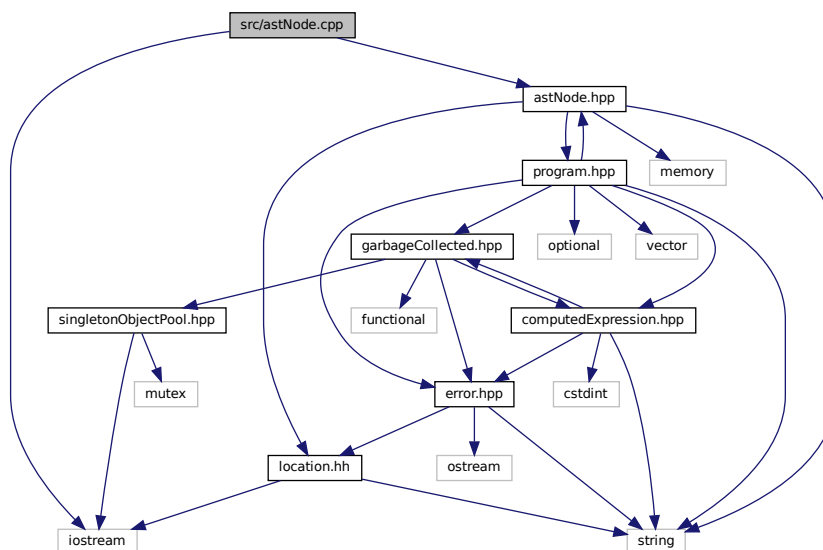
## 6.24 src/astNode.cpp File Reference

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

```
#include <iostream>
```

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

Include dependency graph for astNode.cpp:



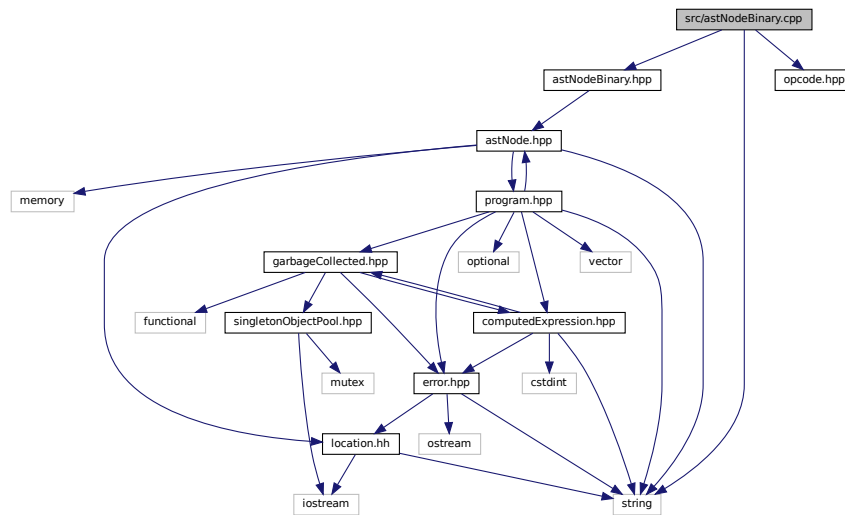
### 6.24.1 Detailed Description

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

## 6.25 src/astNodeBinary.cpp File Reference

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

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



### 6.25.1 Detailed Description

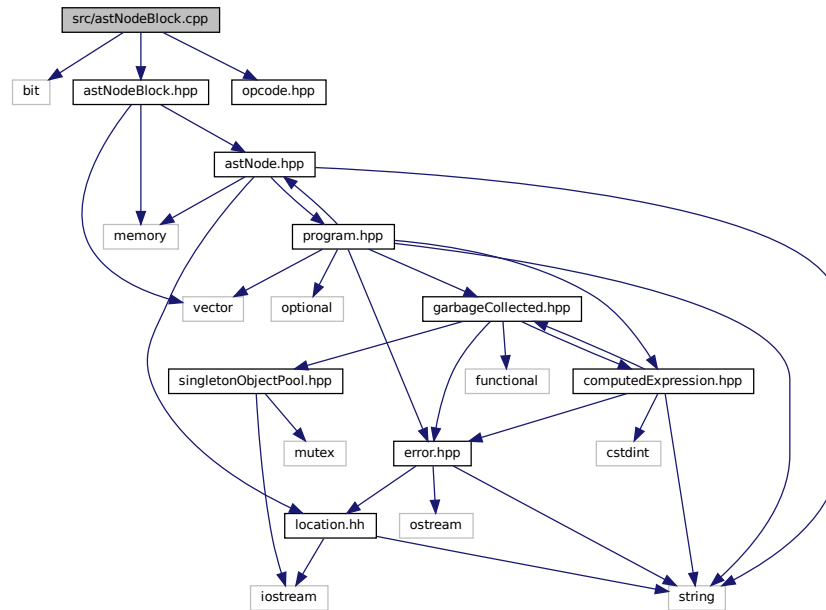
Define the [Tang::AstNodeBinary](#) class.

## 6.26 src/astNodeBlock.cpp File Reference

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

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

Include dependency graph for astNodeBlock.cpp:



### 6.26.1 Detailed Description

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

## 6.27 src/astNodeBoolean.cpp File Reference

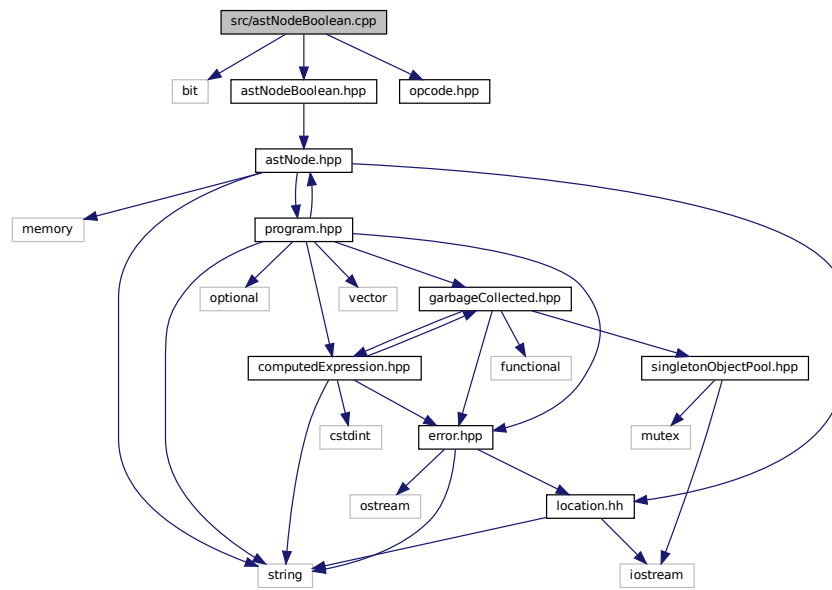
Define the [Tang::AstNodeBoolean](#) class.

```

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

```

Include dependency graph for `astNodeBoolean.cpp`:



### 6.27.1 Detailed Description

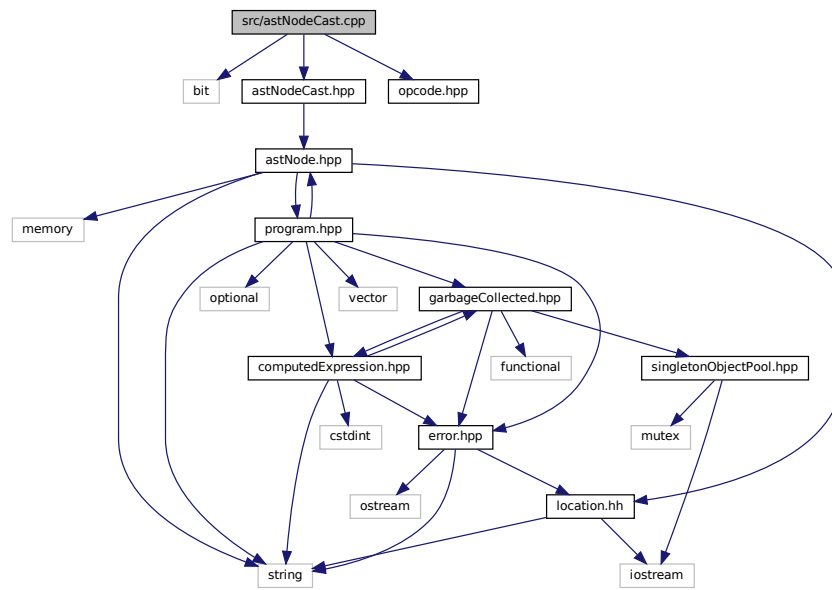
Define the [Tang::AstNodeBoolean](#) class.

## 6.28 src/astNodeCast.cpp File Reference

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

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

Include dependency graph for astNodeCast.cpp:



### 6.28.1 Detailed Description

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

## 6.29 src/astNodeFloat.cpp File Reference

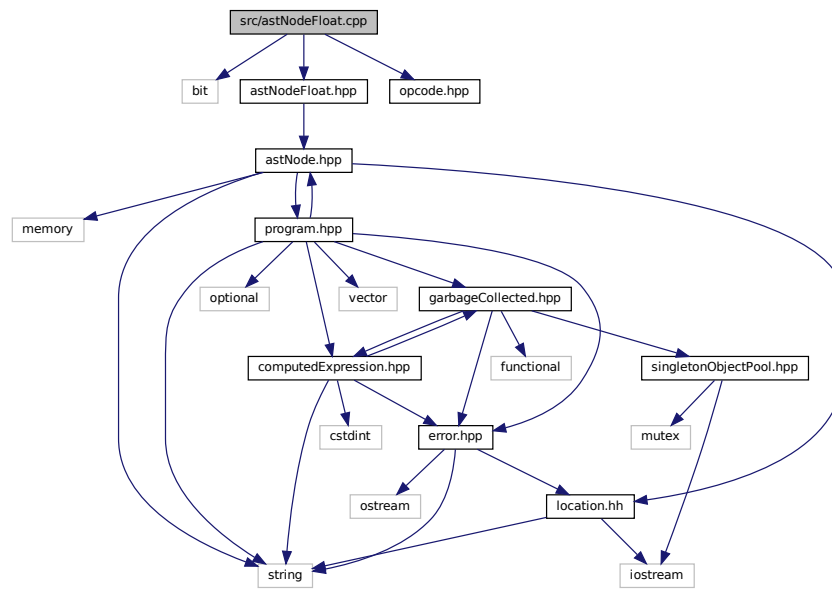
Define the [Tang::AstNodeFloat](#) class.

```

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

```

Include dependency graph for `astNodeFloat.cpp`:



### 6.29.1 Detailed Description

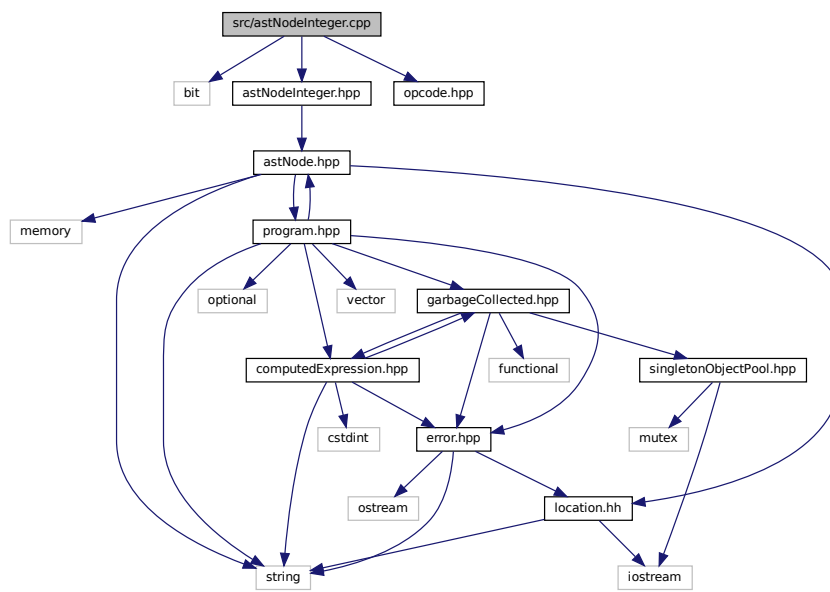
Define the [Tang::AstNodeFloat](#) class.

## 6.30 src/astNodeInteger.cpp File Reference

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

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

Include dependency graph for astNodeInteger.cpp:



### 6.30.1 Detailed Description

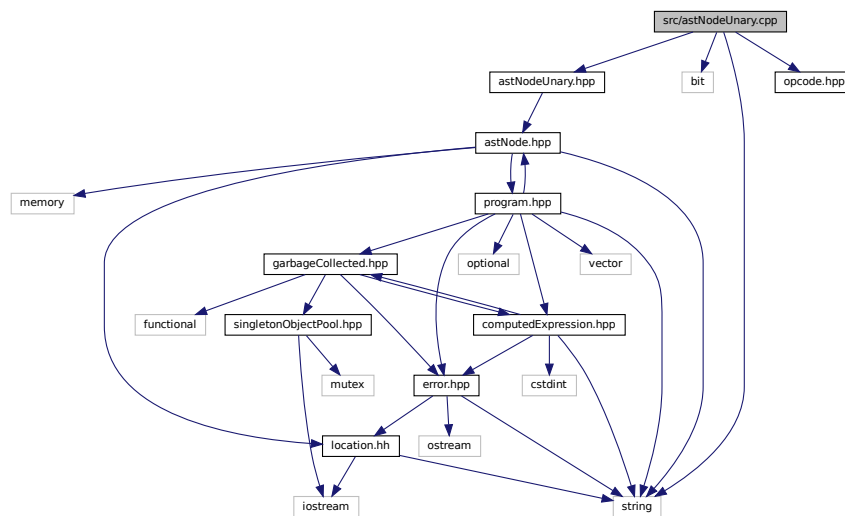
Define the `Tang::AstNodeInteger` class.

### 6.31 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`:



### 6.31.1 Detailed Description

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

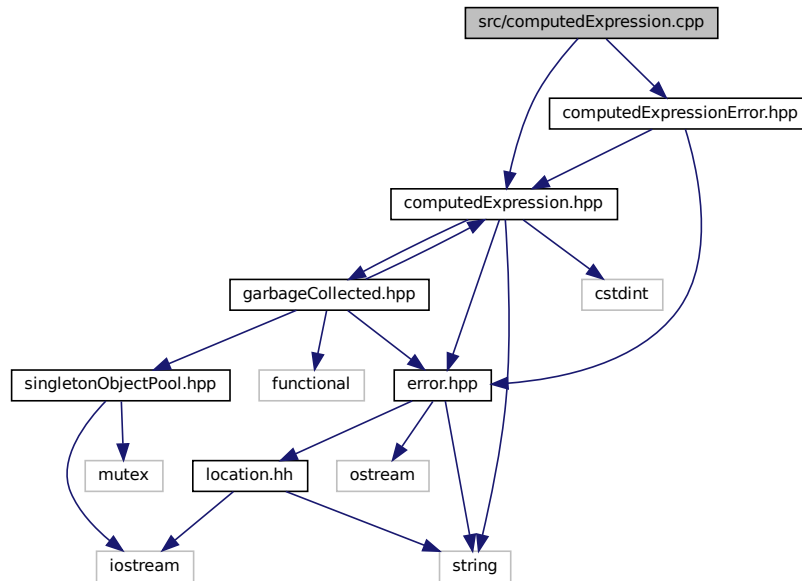
## 6.32 src/computedExpression.cpp File Reference

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

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



Include dependency graph for computedExpression.cpp:



### 6.32.1 Detailed Description

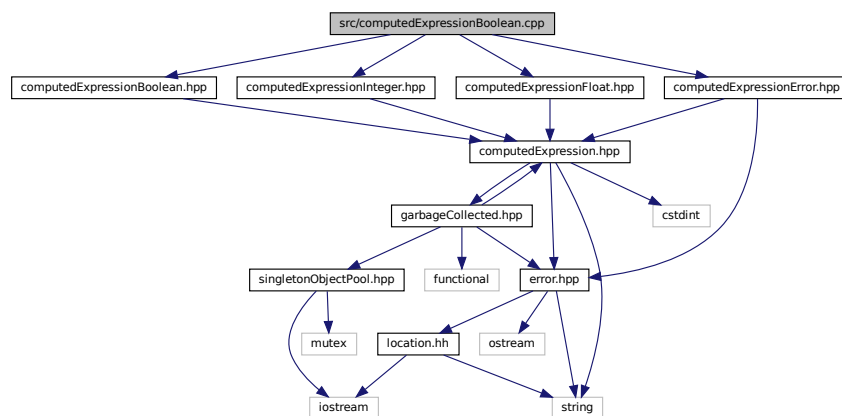
Define the [Tang::ComputedExpression](#) class.

## 6.33 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:



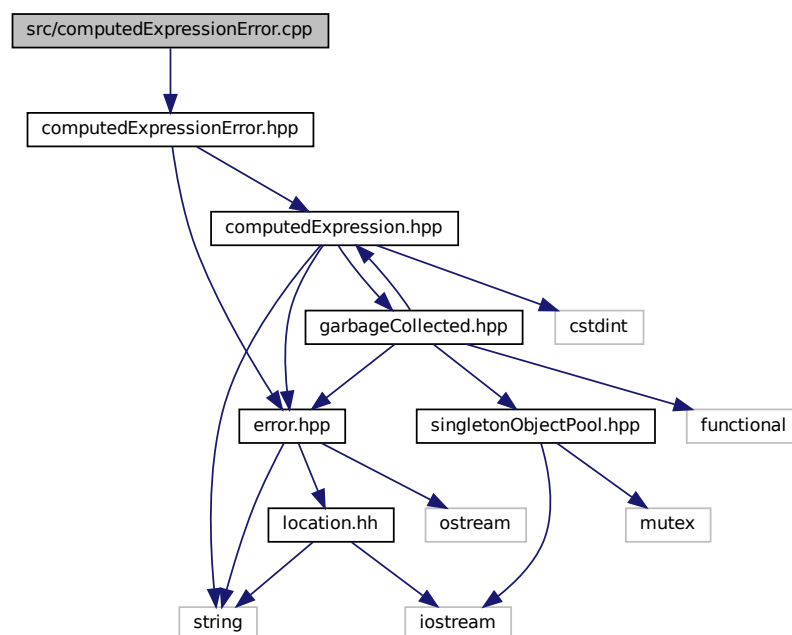
### 6.33.1 Detailed Description

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

## 6.34 src/computedExpressionError.cpp File Reference

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

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



### 6.34.1 Detailed Description

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

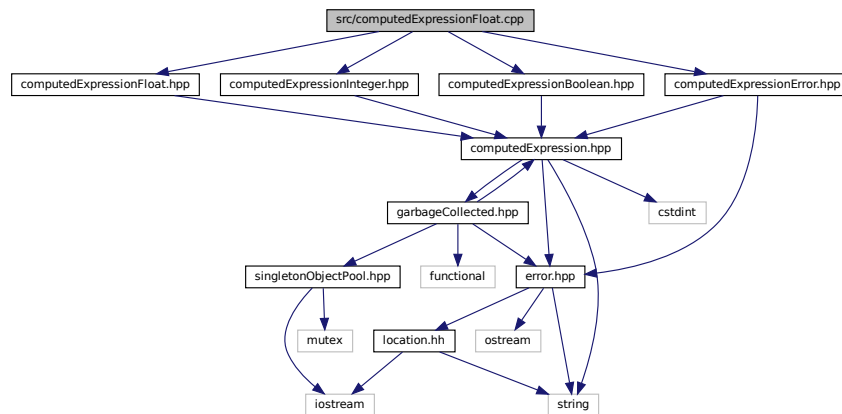
## 6.35 src/computedExpressionFloat.cpp File Reference

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

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

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

Include dependency graph for computedExpressionFloat.cpp:



### 6.35.1 Detailed Description

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

## 6.36 src/computedExpressionInteger.cpp File Reference

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

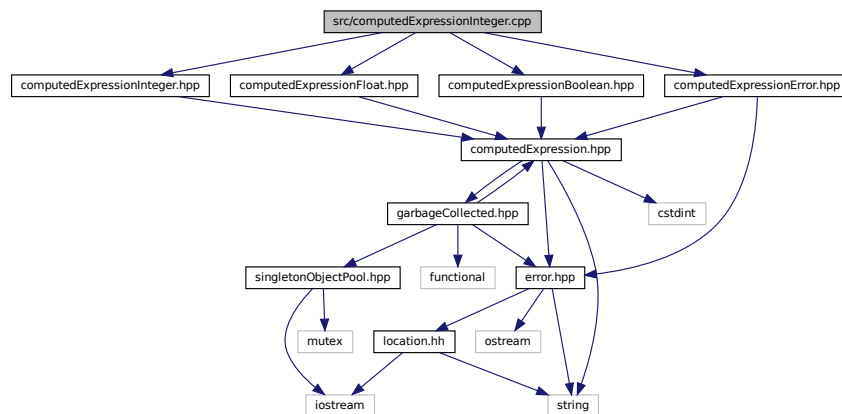
```
#include "computedExpressionInteger.hpp"
```

```
#include "computedExpressionFloat.hpp"
```

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

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

Include dependency graph for computedExpressionInteger.cpp:



### 6.36.1 Detailed Description

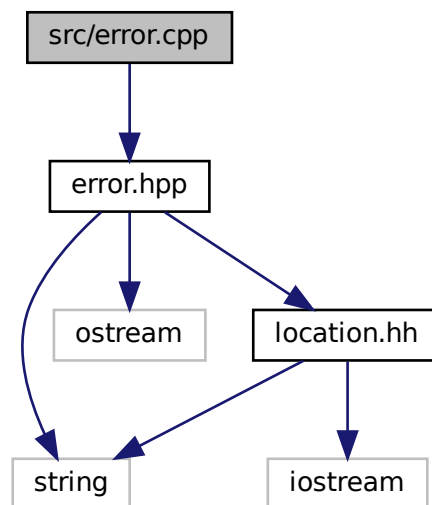
Define the [Tang::ComputedExpressionInteger](#) class.

## 6.37 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)`

### 6.37.1 Detailed Description

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

### 6.37.2 Function Documentation

#### 6.37.2.1 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

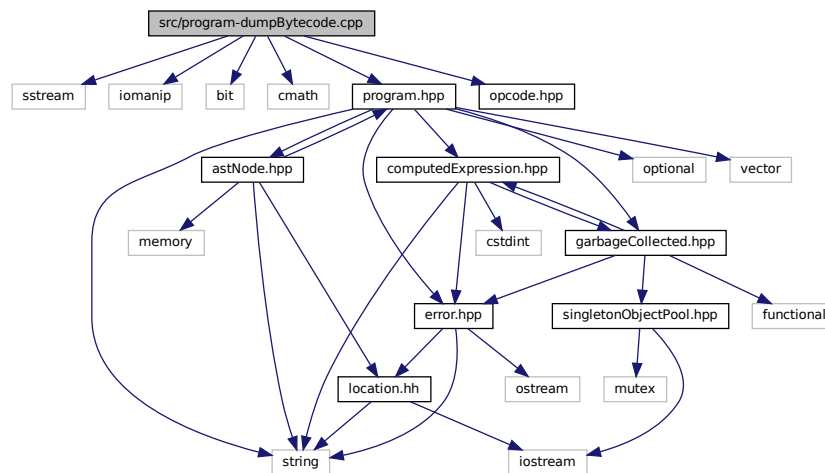
The output stream.

## 6.38 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.

#### 6.38.1 Detailed Description

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

## 6.38.2 Macro Definition Documentation

### 6.38.2.1 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

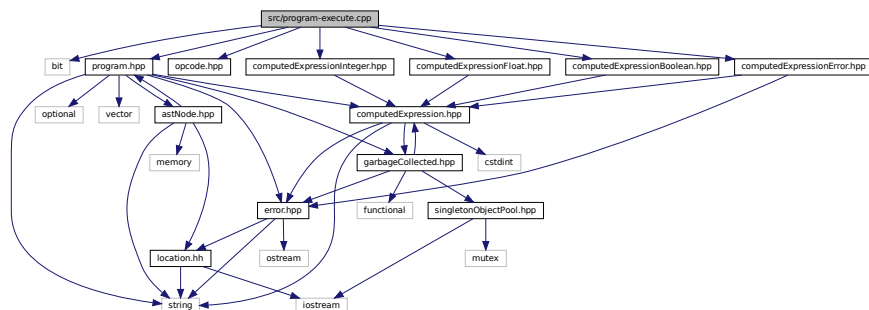
x	The number of additional vector entries that should exist.
---	--

## 6.39 src/program-execute.cpp File Reference

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

```
#include <bit>  
#include "program.hpp"  
#include "opcode.hpp"  
#include "computedExpressionError.hpp"  
#include "computedExpressionInteger.hpp"  
#include "computedExpressionFloat.hpp"  
#include "computedExpressionBoolean.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.*

### 6.39.1 Detailed Description

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

### 6.39.2 Macro Definition Documentation

#### 6.39.2.1 EXECUTEPROGRAMCHECK

```
#define EXECUTEPROGRAMCHECK(  
    x )
```

##### Value:

```
if (this->bytecode.size() < (pc + (x))) { \
    stack.push_back(GarbageCollected::make<ComputedExpressionError>(Error{"Opcode instruction  
truncated."})); \
    pc = this->bytecode.size(); \
    break; \
}
```

Verify the size of the Bytecode vector so that it may be safely accessed.

##### Parameters

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

#### 6.39.2.2 STACKCHECK

```
#define STACKCHECK(  
    x )
```

##### Value:

```
if (stack.size() < (fp + (x))) { \
    stack.push_back(GarbageCollected::make<ComputedExpressionError>(Error{"Insufficient stack depth."})); \
    pc = this->bytecode.size(); \
    break; \
}
```

Verify the size of the stack vector so that it may be safely accessed.

##### Parameters

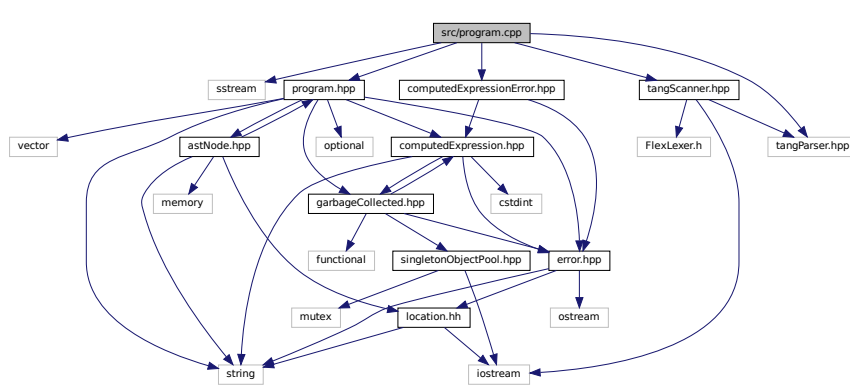
<code>x</code>	The number of entries that should exist in the stack.
----------------	---

## 6.40 src/program.cpp File Reference

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

```
#include <sstream>
#include "program.hpp"
#include "tangScanner.hpp"
#include "tangParser.hpp"
#include "computedExpressionError.hpp"
```

Include dependency graph for program.cpp:



### 6.40.1 Detailed Description

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

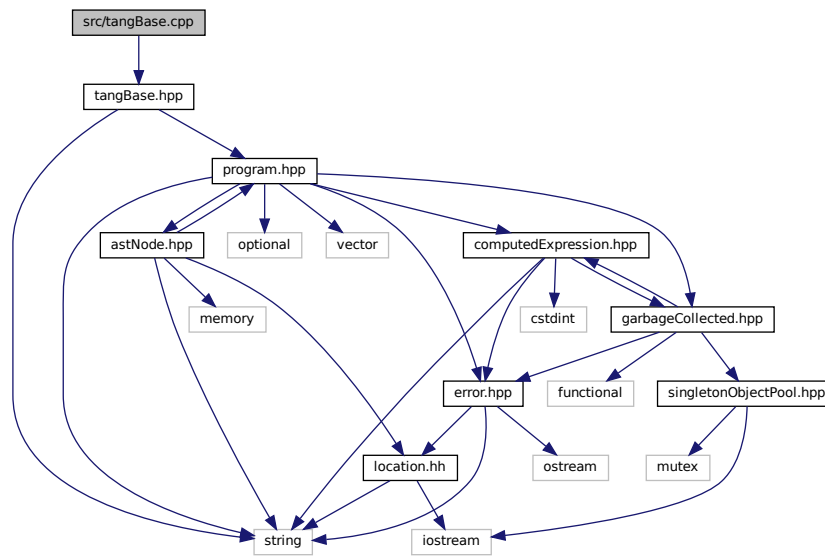
## 6.41 src/tangBase.cpp File Reference

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



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

Include dependency graph for tangBase.cpp:



### 6.41.1 Detailed Description

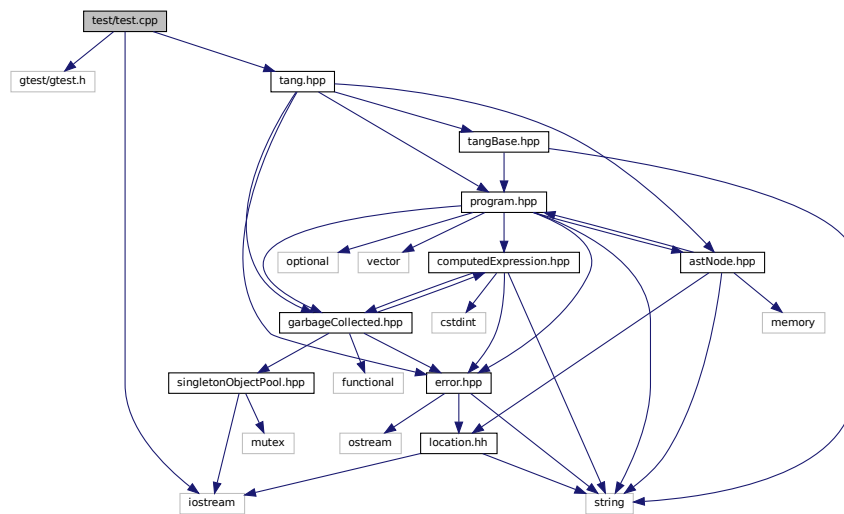
Define the [Tang::TangBase](#) class.

## 6.42 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, Integer)
- **TEST** (Declare, Float)
- **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, Boolean)
- **TEST** (Expression, Not)
- **TEST** (Expression, LessThan)
- **TEST** (Expression, LessThanEqual)
- **TEST** (Expression, GreaterThan)
- **TEST** (Expression, GreaterThanEqual)
- **TEST** (Expression, Equal)
- **TEST** (Expression, NotEqual)
- **TEST** (CodeBlock, Statements)
- **int main** (int argc, char \*\*argv)

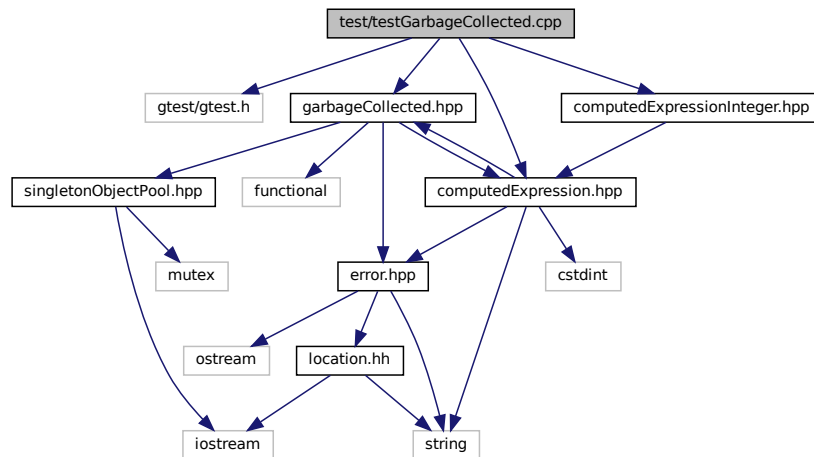
### 6.42.1 Detailed Description

Test the general language behaviors.

## 6.43 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)

#### 6.43.1 Detailed Description

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

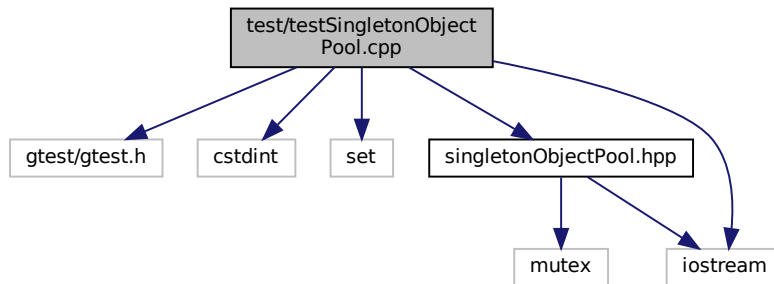
## 6.44 test/testSingletonObjectPool.cpp File Reference

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

```
#include <gtest/gtest.h>
#include <cstdint>
#include <set>
#include "singletonObjectPool.hpp"
```

```
#include <iostream>
```

Include dependency graph for testSingletonObjectPool.cpp:



## Functions

- **TEST** (Singleton, SameForSameType)
- **TEST** (Singleton, DifferentForDifferentTypes)
- **TEST** (Get, SuccessiveCallsProduceDifferentMemoryAddresses)
- **TEST** (Recycle, RecycledObjectsIsReused)
- **TEST** (Get, SuccessiveCallsAreSequential)
- **TEST** (Get, KeepsGeneratingDifferentPointers)
- **TEST** (Recycle, WorksAfterLargeNumberOfAllocations)
- `int main` (int argc, char \*\*argv)

### 6.44.1 Detailed Description

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

# Index

- \_\_add
  - Tang::ComputedExpression, [34](#)
  - Tang::ComputedExpressionBoolean, [46](#)
  - Tang::ComputedExpressionError, [54](#)
  - Tang::ComputedExpressionFloat, [63](#)
  - Tang::ComputedExpressionInteger, [71](#)
- \_\_boolean
  - Tang::ComputedExpression, [36](#)
  - Tang::ComputedExpressionBoolean, [46](#)
  - Tang::ComputedExpressionError, [55](#)
  - Tang::ComputedExpressionFloat, [63](#)
  - Tang::ComputedExpressionInteger, [72](#)
- \_\_divide
  - Tang::ComputedExpression, [36](#)
  - Tang::ComputedExpressionBoolean, [46](#)
  - Tang::ComputedExpressionError, [55](#)
  - Tang::ComputedExpressionFloat, [63](#)
  - Tang::ComputedExpressionInteger, [72](#)
- \_\_equal
  - Tang::ComputedExpression, [36](#)
  - Tang::ComputedExpressionBoolean, [47](#)
  - Tang::ComputedExpressionError, [55](#)
  - Tang::ComputedExpressionFloat, [64](#)
  - Tang::ComputedExpressionInteger, [72](#)
- \_\_float
  - Tang::ComputedExpression, [38](#)
  - Tang::ComputedExpressionBoolean, [47](#)
  - Tang::ComputedExpressionError, [56](#)
  - Tang::ComputedExpressionFloat, [64](#)
  - Tang::ComputedExpressionInteger, [73](#)
- \_\_integer
  - Tang::ComputedExpression, [38](#)
  - Tang::ComputedExpressionBoolean, [47](#)
  - Tang::ComputedExpressionError, [56](#)
  - Tang::ComputedExpressionFloat, [64](#)
  - Tang::ComputedExpressionInteger, [73](#)
- \_\_lessThan
  - Tang::ComputedExpression, [38](#)
  - Tang::ComputedExpressionBoolean, [48](#)
  - Tang::ComputedExpressionError, [56](#)
  - Tang::ComputedExpressionFloat, [65](#)
  - Tang::ComputedExpressionInteger, [73](#)
- \_\_modulo
  - Tang::ComputedExpression, [39](#)
  - Tang::ComputedExpressionBoolean, [48](#)
  - Tang::ComputedExpressionError, [57](#)
  - Tang::ComputedExpressionFloat, [65](#)
  - Tang::ComputedExpressionInteger, [74](#)
- \_\_multiply
  - Tang::ComputedExpression, [39](#)
  - Tang::ComputedExpressionBoolean, [48](#)
  - Tang::ComputedExpressionError, [57](#)
  - Tang::ComputedExpressionFloat, [65](#)
  - Tang::ComputedExpressionInteger, [74](#)
- \_\_negative
  - Tang::ComputedExpression, [39](#)
  - Tang::ComputedExpressionBoolean, [49](#)
  - Tang::ComputedExpressionError, [57](#)
  - Tang::ComputedExpressionFloat, [66](#)
  - Tang::ComputedExpressionInteger, [74](#)
- \_\_not
  - Tang::ComputedExpression, [40](#)
  - Tang::ComputedExpressionBoolean, [49](#)
  - Tang::ComputedExpressionError, [58](#)
  - Tang::ComputedExpressionFloat, [66](#)
  - Tang::ComputedExpressionInteger, [75](#)
- \_\_subtract
  - Tang::ComputedExpression, [40](#)
  - Tang::ComputedExpressionBoolean, [49](#)
  - Tang::ComputedExpressionError, [58](#)
  - Tang::ComputedExpressionFloat, [66](#)
  - Tang::ComputedExpressionInteger, [75](#)
- ~GarbageCollected
  - Tang::GarbageCollected, [84](#)
- ADD
  - opcode.hpp, [127](#)
- Add
  - Tang::AstNodeBinary, [14](#)
- addBytecode
  - Tang::Program, [100](#)
- AstNode
  - Tang::AstNode, [11](#)
- AstNodeBinary
  - Tang::AstNodeBinary, [14](#)
- AstNodeBlock
  - Tang::AstNodeBlock, [17](#)
- AstNodeBoolean
  - Tang::AstNodeBoolean, [20](#)
- AstNodeCast
  - Tang::AstNodeCast, [23](#)
- AstNodeFloat
  - Tang::AstNodeFloat, [26](#)
- AstNodeInteger
  - Tang::AstNodeInteger, [29](#)
- AstNodeUnary
  - Tang::AstNodeUnary, [32](#)
- BOOLEAN

- opcode.hpp, 127
- Boolean
  - Tang::AstNodeCast, 23
- build/generated/location.hh, 109
- CASTBOOLEAN
  - opcode.hpp, 127
- CASTFLOAT
  - opcode.hpp, 127
- CASTINTEGER
  - opcode.hpp, 127
- CodeType
  - Tang::Program, 100
- compileScript
  - Tang::TangBase, 104
- ComputedExpressionBoolean
  - Tang::ComputedExpressionBoolean, 44
- ComputedExpressionError
  - Tang::ComputedExpressionError, 54
- ComputedExpressionFloat
  - Tang::ComputedExpressionFloat, 62
- ComputedExpressionInteger
  - Tang::ComputedExpressionInteger, 71
- DIVIDE
  - opcode.hpp, 127
- Divide
  - Tang::AstNodeBinary, 14
- dump
  - Tang::ComputedExpression, 40
  - Tang::ComputedExpressionBoolean, 50
  - Tang::ComputedExpressionError, 58
  - Tang::ComputedExpressionFloat, 67
  - Tang::ComputedExpressionInteger, 75
- dumpBytecode
  - Tang::Program, 101
- DUMPPROGRAMCHECK
  - program-dumpBytecode.cpp, 146
- EQ
  - opcode.hpp, 127
- Equal
  - Tang::AstNodeBinary, 14
- Error
  - Tang::Error, 80
- error.cpp
  - operator<<, 144
- execute
  - Tang::Program, 101
- EXECUTEPROGRAMCHECK
  - program-execute.cpp, 147
- FLOAT
  - opcode.hpp, 127
- Float
  - Tang::AstNodeCast, 23
- GarbageCollected
  - Tang::GarbageCollected, 83, 84
- get
  - Tang::SingletonObjectPool< T >, 103
- get\_next\_token
  - Tang::TangScanner, 106
- getAst
  - Tang::Program, 101
- getCode
  - Tang::Program, 101
- getInstance
  - Tang::SingletonObjectPool< T >, 103
- getResult
  - Tang::Program, 102
- GreaterThan
  - Tang::AstNodeBinary, 14
- GreaterThanEqual
  - Tang::AstNodeBinary, 14
- GT
  - opcode.hpp, 127
- GTE
  - opcode.hpp, 127
- include/astNode.hpp, 111
- include/astNodeBinary.hpp, 112
- include/astNodeBlock.hpp, 113
- include/astNodeBoolean.hpp, 114
- include/astNodeCast.hpp, 115
- include/astNodeFloat.hpp, 116
- include/astNodeInteger.hpp, 117
- include/astNodeUnary.hpp, 118
- include/computedExpression.hpp, 119
- include/computedExpressionBoolean.hpp, 120
- include/computedExpressionError.hpp, 121
- include/computedExpressionFloat.hpp, 122
- include/computedExpressionInteger.hpp, 123
- include/error.hpp, 124
- include/garbageCollected.hpp, 125
- include/macros.hpp, 126
- include/opcode.hpp, 126
- include/program.hpp, 128
- include/singletonObjectPool.hpp, 129
- include/tang.hpp, 130
- include/tangBase.hpp, 131
- include/tangScanner.hpp, 132
- INTEGER
  - opcode.hpp, 127
- Integer
  - Tang::AstNodeCast, 23
- is\_equal
  - Tang::ComputedExpression, 41, 42
  - Tang::ComputedExpressionBoolean, 50, 51
  - Tang::ComputedExpressionError, 58–60
  - Tang::ComputedExpressionFloat, 67, 68
  - Tang::ComputedExpressionInteger, 76, 77
- LessThan
  - Tang::AstNodeBinary, 14
- LessThanEqual
  - Tang::AstNodeBinary, 14
- location.hh

- operator<<, [110](#), [111](#)
- LT
  - opcode.hpp, [127](#)
- LTE
  - opcode.hpp, [127](#)
- macros.hpp
  - TANG\_UNUSED, [126](#)
- make
  - Tang::GarbageCollected, [84](#)
- makeCopy
  - Tang::AstNode, [11](#)
  - Tang::AstNodeBinary, [15](#)
  - Tang::AstNodeBlock, [17](#)
  - Tang::AstNodeBoolean, [20](#)
  - Tang::AstNodeCast, [24](#)
  - Tang::AstNodeFloat, [26](#)
  - Tang::AstNodeInteger, [29](#)
  - Tang::AstNodeUnary, [33](#)
  - Tang::ComputedExpression, [42](#)
  - Tang::ComputedExpressionBoolean, [51](#)
  - Tang::ComputedExpressionError, [60](#)
  - Tang::ComputedExpressionFloat, [68](#)
  - Tang::ComputedExpressionInteger, [77](#)
- MODULO
  - opcode.hpp, [127](#)
- Modulo
  - Tang::AstNodeBinary, [14](#)
- MULTIPLY
  - opcode.hpp, [127](#)
- Multiply
  - Tang::AstNodeBinary, [14](#)
- NEGATIVE
  - opcode.hpp, [127](#)
- Negative
  - Tang::AstNodeUnary, [32](#)
- NEQ
  - opcode.hpp, [127](#)
- NOT
  - opcode.hpp, [127](#)
- Not
  - Tang::AstNodeUnary, [32](#)
- NotEqual
  - Tang::AstNodeBinary, [14](#)
- Opcode
  - opcode.hpp, [127](#)
- opcode.hpp
  - ADD, [127](#)
  - BOOLEAN, [127](#)
  - CASTBOOLEAN, [127](#)
  - CASTFLOAT, [127](#)
  - CASTINTEGER, [127](#)
  - DIVIDE, [127](#)
  - EQ, [127](#)
  - FLOAT, [127](#)
  - GT, [127](#)
  - GTE, [127](#)
  - INTEGER, [127](#)
  - LT, [127](#)
  - LTE, [127](#)
  - MODULO, [127](#)
  - MULTIPLY, [127](#)
  - NEGATIVE, [127](#)
  - NEQ, [127](#)
  - NOT, [127](#)
  - Opcode, [127](#)
  - POP, [127](#)
  - SUBTRACT, [127](#)
- Operation
  - Tang::AstNodeBinary, [14](#)
- Operator
  - Tang::AstNodeUnary, [32](#)
- operator!
  - Tang::GarbageCollected, [85](#)
- operator!=
  - Tang::GarbageCollected, [85](#)
- operator<
  - Tang::GarbageCollected, [90](#)
- operator<<
  - error.cpp, [144](#)
  - location.hh, [110](#), [111](#)
  - Tang::Error, [80](#)
  - Tang::GarbageCollected, [95](#)
- operator<=
  - Tang::GarbageCollected, [90](#)
- operator>
  - Tang::GarbageCollected, [94](#)
- operator>=
  - Tang::GarbageCollected, [94](#)
- operator\*
  - Tang::GarbageCollected, [86](#), [87](#)
- operator+
  - Tang::GarbageCollected, [87](#)
- operator-
  - Tang::GarbageCollected, [88](#)
- operator->
  - Tang::GarbageCollected, [89](#)
- operator/
  - Tang::GarbageCollected, [89](#)
- operator=
  - Tang::GarbageCollected, [91](#)
- operator==
  - Tang::GarbageCollected, [92](#), [93](#)
- operator%
  - Tang::GarbageCollected, [86](#)
- POP
  - opcode.hpp, [127](#)
- Program
  - Tang::Program, [100](#)
- program-dumpBytecode.cpp
  - DUMPPROGRAMCHECK, [146](#)
- program-execute.cpp
  - EXECUTEPROGRAMCHECK, [147](#)
  - STACKCHECK, [147](#)

- recycle
  - Tang::SingletonObjectPool< T >, 103
- Script
  - Tang::Program, 100
- src/astNode.cpp, 133
- src/astNodeBinary.cpp, 134
- src/astNodeBlock.cpp, 134
- src/astNodeBoolean.cpp, 135
- src/astNodeCast.cpp, 136
- src/astNodeFloat.cpp, 137
- src/astNodeInteger.cpp, 138
- src/astNodeUnary.cpp, 139
- src/computedExpression.cpp, 140
- src/computedExpressionBoolean.cpp, 141
- src/computedExpressionError.cpp, 142
- src/computedExpressionFloat.cpp, 142
- src/computedExpressionInteger.cpp, 143
- src/error.cpp, 144
- src/program-dumpBytecode.cpp, 145
- src/program-execute.cpp, 146
- src/program.cpp, 148
- src/tangBase.cpp, 148
- STACKCHECK
  - program-execute.cpp, 147
- SUBTRACT
  - opcode.hpp, 127
- Subtract
  - Tang::AstNodeBinary, 14
- Tang::AstNode, 9
  - AstNode, 11
  - makeCopy, 11
- Tang::AstNodeBinary, 12
  - Add, 14
  - AstNodeBinary, 14
  - Divide, 14
  - Equal, 14
  - GreaterThan, 14
  - GreaterThanEqual, 14
  - LessThan, 14
  - LessThanEqual, 14
  - makeCopy, 15
  - Modulo, 14
  - Multiply, 14
  - NotEqual, 14
  - Operation, 14
  - Subtract, 14
- Tang::AstNodeBlock, 15
  - AstNodeBlock, 17
  - makeCopy, 17
- Tang::AstNodeBoolean, 18
  - AstNodeBoolean, 20
  - makeCopy, 20
- Tang::AstNodeCast, 21
  - AstNodeCast, 23
  - Boolean, 23
  - Float, 23
  - Integer, 23
  - makeCopy, 24
  - Type, 23
- Tang::AstNodeFloat, 24
  - AstNodeFloat, 26
  - makeCopy, 26
- Tang::AstNodeInteger, 27
  - AstNodeInteger, 29
  - makeCopy, 29
- Tang::AstNodeUnary, 30
  - AstNodeUnary, 32
  - makeCopy, 33
  - Negative, 32
  - Not, 32
  - Operator, 32
- Tang::ComputedExpression, 33
  - \_\_add, 34
  - \_\_boolean, 36
  - \_\_divide, 36
  - \_\_equal, 36
  - \_\_float, 38
  - \_\_integer, 38
  - \_\_lessThan, 38
  - \_\_modulo, 39
  - \_\_multiply, 39
  - \_\_negative, 39
  - \_\_not, 40
  - \_\_subtract, 40
  - dump, 40
  - is\_equal, 41, 42
  - makeCopy, 42
- Tang::ComputedExpressionBoolean, 43
  - \_\_add, 46
  - \_\_boolean, 46
  - \_\_divide, 46
  - \_\_equal, 47
  - \_\_float, 47
  - \_\_integer, 47
  - \_\_lessThan, 48
  - \_\_modulo, 48
  - \_\_multiply, 48
  - \_\_negative, 49
  - \_\_not, 49
  - \_\_subtract, 49
  - ComputedExpressionBoolean, 44
  - dump, 50
  - is\_equal, 50, 51
  - makeCopy, 51
- Tang::ComputedExpressionError, 52
  - \_\_add, 54
  - \_\_boolean, 55
  - \_\_divide, 55
  - \_\_equal, 55
  - \_\_float, 56
  - \_\_integer, 56
  - \_\_lessThan, 56
  - \_\_modulo, 57
  - \_\_multiply, 57
  - \_\_negative, 57



- \_\_not, 58
  - \_\_subtract, 58
- ComputedExpressionError, 54
- dump, 58
- is\_equal, 58–60
- makeCopy, 60
- Tang::ComputedExpressionFloat, 61
  - \_\_add, 63
  - \_\_boolean, 63
  - \_\_divide, 63
  - \_\_equal, 64
  - \_\_float, 64
  - \_\_integer, 64
  - \_\_lessThan, 65
  - \_\_modulo, 65
  - \_\_multiply, 65
  - \_\_negative, 66
  - \_\_not, 66
  - \_\_subtract, 66
- ComputedExpressionFloat, 62
- dump, 67
- is\_equal, 67, 68
- makeCopy, 68
- Tang::ComputedExpressionInteger, 69
  - \_\_add, 71
  - \_\_boolean, 72
  - \_\_divide, 72
  - \_\_equal, 72
  - \_\_float, 73
  - \_\_integer, 73
  - \_\_lessThan, 73
  - \_\_modulo, 74
  - \_\_multiply, 74
  - \_\_negative, 74
  - \_\_not, 75
  - \_\_subtract, 75
- ComputedExpressionInteger, 71
- dump, 75
- is\_equal, 76, 77
- makeCopy, 77
- Tang::Error, 78
  - Error, 80
- operator<<, 80
- Tang::GarbageCollected, 81
  - ~GarbageCollected, 84
- GarbageCollected, 83, 84
- make, 84
- operator!, 85
- operator!=, 85
- operator<, 90
- operator<<, 95
- operator<=, 90
- operator>, 94
- operator>=, 94
- operator\*, 86, 87
- operator+, 87
- operator-, 88
- operator->, 89
- operator/, 89
- operator=, 91
- operator==, 92, 93
- operator%, 86
- Tang::location, 95
- Tang::position, 97
- Tang::Program, 98
  - addBytecode, 100
  - CodeType, 100
  - dumpBytecode, 101
  - execute, 101
  - getAst, 101
  - getCode, 101
  - getResult, 102
  - Program, 100
  - Script, 100
  - Template, 100
- Tang::SingletonObjectPool< T >, 102
  - get, 103
  - getInstance, 103
  - recycle, 103
- Tang::TangBase, 104
  - compileScript, 104
  - TangBase, 104
- Tang::TangScanner, 105
  - get\_next\_token, 106
  - TangScanner, 106
- TANG\_UNUSED
  - macros.hpp, 126
- TangBase
  - Tang::TangBase, 104
- TangScanner
  - Tang::TangScanner, 106
- Template
  - Tang::Program, 100
- test/test.cpp, 149
- test/testGarbageCollected.cpp, 151
- test/testSingletonObjectPool.cpp, 151
- Type
  - Tang::AstNodeCast, 23