

Tang

0.1

Generated by Doxygen 1.9.1

1 Tang: A Template Language	1
1.1 Quick Description	1
1.2 Features	1
1.3 License	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Class Documentation	9
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::AstNodeFloat Class Reference	12
5.2.1 Detailed Description	14
5.2.2 Constructor & Destructor Documentation	14
5.2.2.1 AstNodeFloat()	14
5.2.3 Member Function Documentation	14
5.2.3.1 makeCopy()	14
5.3 Tang::AstNodeInteger Class Reference	15
5.3.1 Detailed Description	17
5.3.2 Constructor & Destructor Documentation	17
5.3.2.1 AstNodeInteger()	17
5.3.3 Member Function Documentation	17
5.3.3.1 makeCopy()	17
5.4 Tang::ComputedExpression Class Reference	18
5.4.1 Detailed Description	18
5.4.2 Member Function Documentation	18
5.4.2.1 dump()	18
5.4.2.2 is_equal() [1/2]	19
5.4.2.3 is_equal() [2/2]	19
5.4.2.4 makeCopy()	19
5.5 Tang::ComputedExpressionFloat Class Reference	20
5.5.1 Detailed Description	21
5.5.2 Constructor & Destructor Documentation	21
5.5.2.1 ComputedExpressionFloat()	21

5.5.3 Member Function Documentation	21
5.5.3.1 dump()	21
5.5.3.2 is_equal() [1/2]	21
5.5.3.3 is_equal() [2/2]	22
5.5.3.4 makeCopy()	22
5.6 Tang::ComputedExpressionInteger Class Reference	23
5.6.1 Detailed Description	24
5.6.2 Constructor & Destructor Documentation	24
5.6.2.1 ComputedExpressionInteger()	24
5.6.3 Member Function Documentation	24
5.6.3.1 dump()	24
5.6.3.2 is_equal() [1/2]	24
5.6.3.3 is_equal() [2/2]	25
5.6.3.4 makeCopy()	25
5.7 Tang::Error Class Reference	26
5.7.1 Detailed Description	27
5.7.2 Constructor & Destructor Documentation	27
5.7.2.1 Error()	27
5.8 Tang::GarbageCollected Class Reference	27
5.8.1 Detailed Description	28
5.8.2 Constructor & Destructor Documentation	28
5.8.2.1 GarbageCollected() [1/2]	28
5.8.2.2 GarbageCollected() [2/2]	29
5.8.2.3 ~GarbageCollected()	29
5.8.3 Member Function Documentation	29
5.8.3.1 make()	29
5.8.3.2 operator*()	30
5.8.3.3 operator->()	30
5.8.3.4 operator=() [1/2]	30
5.8.3.5 operator=() [2/2]	31
5.8.3.6 operator==()	31
5.8.4 Friends And Related Function Documentation	32
5.8.4.1 operator<<	32
5.9 Tang::location Class Reference	32
5.9.1 Detailed Description	34
5.10 Tang::position Class Reference	34
5.10.1 Detailed Description	35
5.11 Tang::Program Class Reference	35
5.11.1 Detailed Description	37
5.11.2 Member Enumeration Documentation	37
5.11.2.1 CodeType	37
5.11.3 Constructor & Destructor Documentation	37

5.11.3.1 Program()	37
5.11.4 Member Function Documentation	37
5.11.4.1 addBytecode()	38
5.11.4.2 dumpBytecode()	38
5.11.4.3 execute()	38
5.11.4.4 getAst()	38
5.11.4.5 getCode()	39
5.11.4.6 getResult()	39
5.12 Tang::SingletonObjectPool< T > Class Template Reference	39
5.12.1 Member Function Documentation	39
5.12.1.1 get()	40
5.12.1.2 getInstance()	40
5.12.1.3 recycle()	40
5.13 Tang::TangBase Class Reference	40
5.13.1 Detailed Description	41
5.13.2 Constructor & Destructor Documentation	41
5.13.2.1 TangBase()	41
5.13.3 Member Function Documentation	41
5.13.3.1 compileScript()	41
5.14 Tang::TangScanner Class Reference	42
5.14.1 Detailed Description	43
5.14.2 Constructor & Destructor Documentation	43
5.14.2.1 TangScanner()	43
5.14.3 Member Function Documentation	43
5.14.3.1 get_next_token()	43
6 File Documentation	45
6.1 build/generated/location.hh File Reference	45
6.1.1 Detailed Description	46
6.1.2 Function Documentation	46
6.1.2.1 operator<<() [1/2]	47
6.1.2.2 operator<<() [2/2]	47
6.2 include/ast.hpp File Reference	47
6.2.1 Detailed Description	48
6.3 include/computedExpression.hpp File Reference	49
6.4 include/error.hpp File Reference	50
6.4.1 Detailed Description	50
6.5 include/garbageCollected.hpp File Reference	51
6.6 include/macros.hpp File Reference	51
6.6.1 Detailed Description	52
6.6.2 Macro Definition Documentation	52
6.6.2.1 TANG_UNUSED	52

6.7 include/opcode.hpp File Reference	52
6.7.1 Detailed Description	53
6.7.2 Enumeration Type Documentation	53
6.7.2.1 Opcode	53
6.8 include/program.hpp File Reference	53
6.8.1 Detailed Description	54
6.9 include/singletonObjectPool.hpp File Reference	55
6.10 include/tang.hpp File Reference	56
6.10.1 Detailed Description	56
6.11 include/tangBase.hpp File Reference	56
6.11.1 Detailed Description	57
6.12 include/tangScanner.hpp File Reference	57
6.12.1 Detailed Description	58
6.13 src/ast.cpp File Reference	58
6.14 src/computedExpression.cpp File Reference	59
6.15 src/error.cpp File Reference	59
6.16 src/program.cpp File Reference	60
6.16.1 Macro Definition Documentation	60
6.16.1.1 DUMPPROGRAMCHECK	60
6.16.1.2 EXECUTEPROGRAMCHECK	61
6.17 src/tangBase.cpp File Reference	61
6.18 test/testSingletonObjectPool.cpp File Reference	61
Index	63

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::AstNodeFloat	12
Tang::AstNodeInteger	15
Tang::ComputedExpression	18
Tang::ComputedExpressionFloat	20
Tang::ComputedExpressionInteger	23
Tang::Error	26
Tang::GarbageCollected	27
Tang::location	32
Tang::position	34
Tang::Program	35
Tang::SingletonObjectPool< T >	39
Tang::TangBase	40
TangTangFlexLexer	
Tang::TangScanner	42

Chapter 3

Class Index

3.1 Class List

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

Tang::AstNode	Base class for representing nodes of an Abstract Syntax Tree (AST)	9
Tang::AstNodeFloat	An AstNode that represents an float literal	12
Tang::AstNodeInteger	An AstNode that represents an integer literal	15
Tang::ComputedExpression	Represents the result of a computation that has been executed	18
Tang::ComputedExpressionFloat	Represents a Float that is the result of a computation	20
Tang::ComputedExpressionInteger	Represents an Integer that is the result of a computation	23
Tang::Error	Used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error	26
Tang::GarbageCollected	A container that acts as a resource-counting garbage collector for the specified type	27
Tang::location	Two points in a source file	32
Tang::position	A point in a source file	34
Tang::Program	Represents a compiled script or template that may be executed	35
Tang::SingletonObjectPool< T >	39
Tang::TangBase	The base class for the Tang programming language	40
Tang::TangScanner	The Flex lexer class for the main Tang language	42

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 Tang ::location class	45
include/ast.hpp	
Define the Tang::AstNode and its associated/derivative classes	47
include/computedExpression.hpp	49
include/error.hpp	
Define the Tang::Error class used to describe syntax and runtime errors	50
include/garbageCollected.hpp	51
include/macros.hpp	
Contains generic macros	51
include/opcode.hpp	
Declare the Opcodes used in the Bytecode representation of a program	52
include/program.hpp	
Define the Tang::Program class used to compile and execute source code	53
include/singletonObjectPool.hpp	55
include/tang.hpp	
Header file supplied for use by 3rd party code so that they can easily include all necessary headers	56
include/tangBase.hpp	
Defines the Tang::TangBase class used to interact with Tang	56
include/tangScanner.hpp	
Defines the Tang::TangScanner used to tokenize a Tang script	57
src/ast.cpp	58
src/computedExpression.cpp	59
src/error.cpp	59
src/program.cpp	60
src/tangBase.cpp	61
test/testSingletonObjectPool.cpp	61

Chapter 5

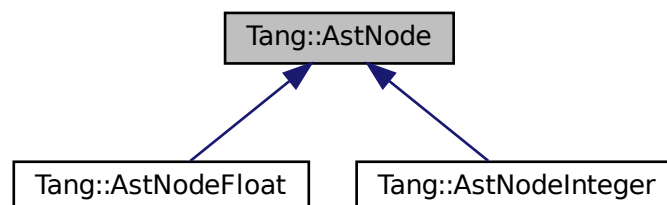
Class Documentation

5.1 Tang::AstNode Class Reference

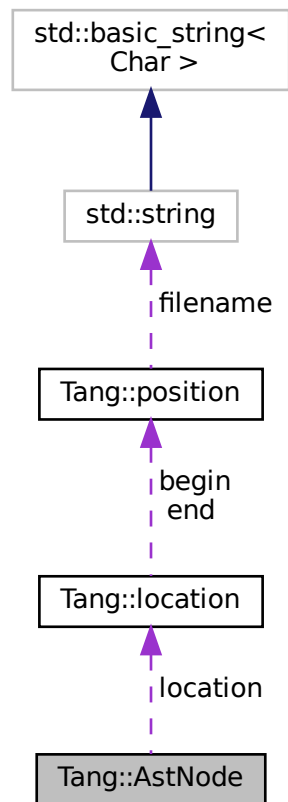
Base class for representing nodes of an Abstract Syntax Tree (AST).

```
#include <ast.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 `AstNode * makeCopy` () const
Provide a copy of the AstNode (recursively, if appropriate).

Protected Member Functions

- `AstNode` (Tang::location loc)
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()

```
Tang::AstNode::AstNode (
    Tang::location loc ) [inline], [protected]
```

The generic constructor.

It should never be called on its own.

Parameters

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

5.1.3 Member Function Documentation

5.1.3.1 makeCopy()

```
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::AstNodeFloat](#), and [Tang::AstNodeInteger](#).

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

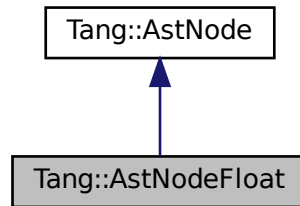
- [include/ast.hpp](#)
- [src/ast.cpp](#)

5.2 Tang::AstNodeFloat Class Reference

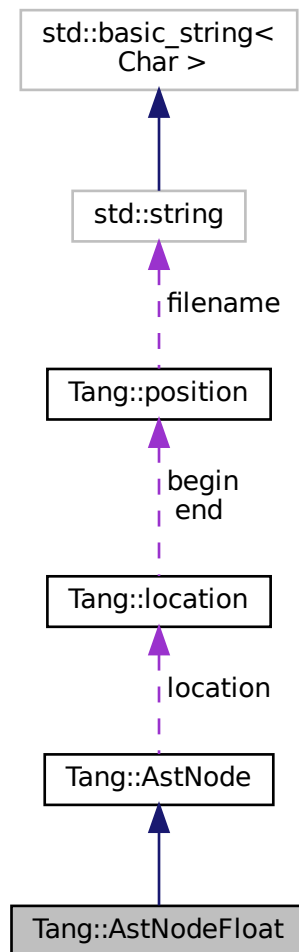
An [AstNode](#) that represents an float literal.

```
#include <ast.hpp>
```

Inheritance diagram for Tang::AstNodeFloat:



Collaboration diagram for Tang::AstNodeFloat:



Public Member Functions

- [AstNodeFloat](#) (double number, [Tang::location](#) loc)
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 [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 an float literal.

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

5.2.2 Constructor & Destructor Documentation

5.2.2.1 AstNodeFloat()

```
Tang::AstNodeFloat::AstNodeFloat (
    double number,
    Tang::location loc ) [inline]
```

The constructor.

Parameters

<i>number</i>	The number to represent.
<i>loc</i>	The location associated with the expression. @location The location associated with this node.

5.2.3 Member Function Documentation

5.2.3.1 makeCopy()

```
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:

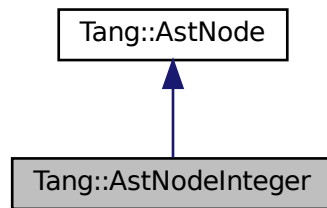
- include/[ast.hpp](#)
- src/[ast.cpp](#)

5.3 Tang::AstNodeInteger Class Reference

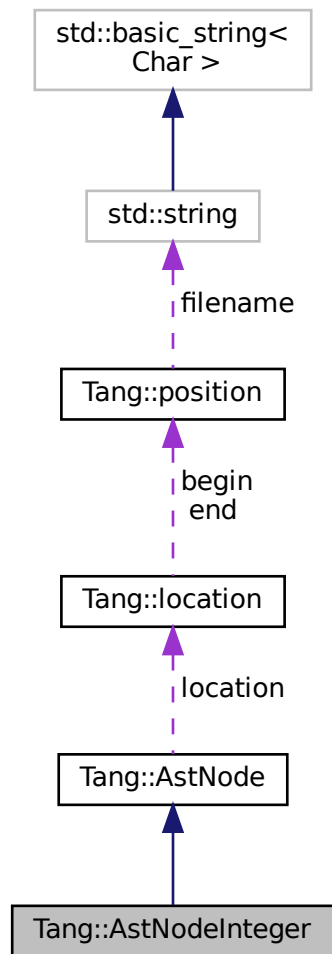
An [AstNode](#) that represents an integer literal.

```
#include <ast.hpp>
```

Inheritance diagram for Tang::AstNodeInteger:



Collaboration diagram for Tang::AstNodeInteger:



Public Member Functions

- `AstNodeInteger` (`int64_t` number, `Tang::location` loc)
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 `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 an integer literal.

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

5.3.2 Constructor & Destructor Documentation

5.3.2.1 AstNodeInteger()

```
Tang::AstNodeInteger::AstNodeInteger (
    int64_t number,
    Tang::location loc ) [inline]
```

The constructor.

Parameters

<i>number</i>	The number to represent.
<i>loc</i>	The location associated with the expression. @location The location associated with this node.

5.3.3 Member Function Documentation

5.3.3.1 makeCopy()

```
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:

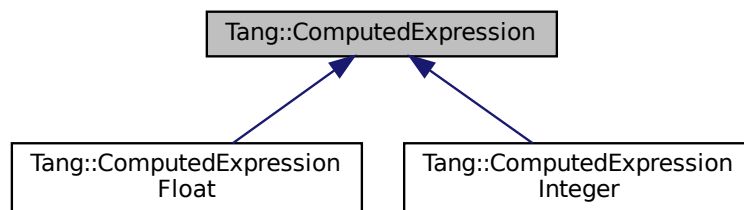
- include/[ast.hpp](#)
- src/[ast.cpp](#)

5.4 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.

5.4.1 Detailed Description

Represents the result of a computation that has been executed.

5.4.2 Member Function Documentation

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

5.4.2.2 is_equal() [1/2]

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

5.4.2.3 is_equal() [2/2]

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

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

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

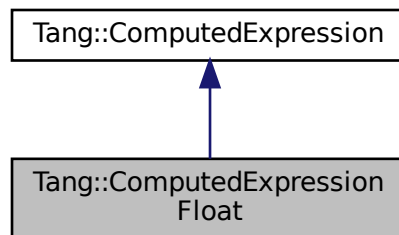
- [include/computedExpression.hpp](#)
- [src/computedExpression.cpp](#)

5.5 Tang::ComputedExpressionFloat Class Reference

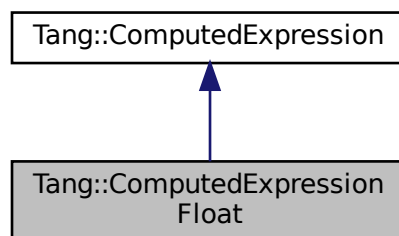
Represents a Float that is the result of a computation.

```
#include <computedExpression.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.

5.5.1 Detailed Description

Represents a Float that is the result of a computation.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 ComputedExpressionFloat()

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

Construct a Float result.

Parameters

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

5.5.3 Member Function Documentation

5.5.3.1 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](#).

5.5.3.2 is_equal() [1/2]

```
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](#).

5.5.3.3 is_equal() [2/2]

```
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](#).

5.5.3.4 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:

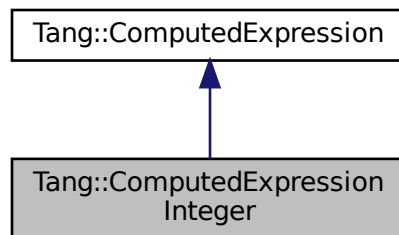
- [include/computedExpression.hpp](#)
- [src/computedExpression.cpp](#)

5.6 Tang::ComputedExpressionInteger Class Reference

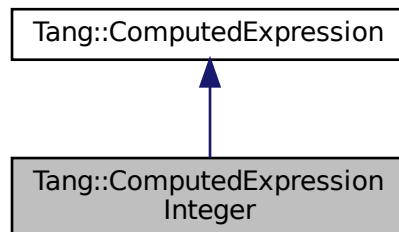
Represents an Integer that is the result of a computation.

```
#include <computedExpression.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.

5.6.1 Detailed Description

Represents an Integer that is the result of a computation.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 ComputedExpressionInteger()

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

Construct an Integer result.

Parameters

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

5.6.3 Member Function Documentation

5.6.3.1 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](#).

5.6.3.2 is_equal() [1/2]

```
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](#).

5.6.3.3 is_equal() [2/2]

```
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](#).

5.6.3.4 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:

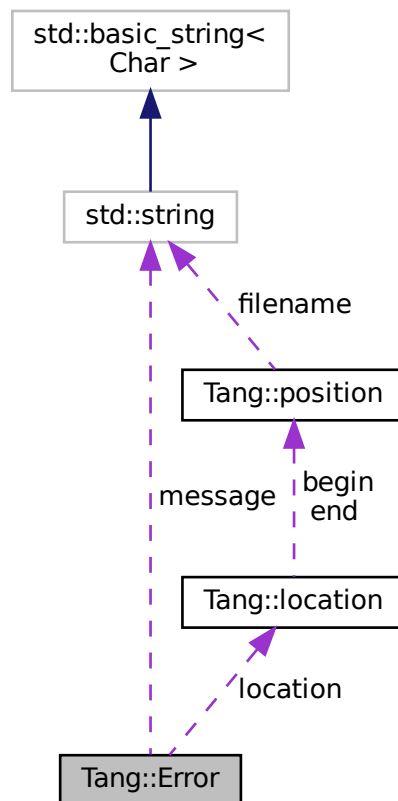
- [include/computedExpression.hpp](#)
- [src/computedExpression.cpp](#)

5.7 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 \(std::string message, Tang::location location\)](#)
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.

5.7.1 Detailed Description

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

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Error()

```
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.

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

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

5.8 Tang::GarbageCollected Class Reference

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

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

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==](#) (auto &val) const
- Compare the [GarbageCollected](#) tracked object with a supplied value.*

Static Public Member Functions

- template<class T , typename... Args>
static [GarbageCollected](#) [make](#) (Args... args)
Creates a garbage-collected object of the specified type.

Friends

- std::ostream & [operator<<](#) (std::ostream &out, const [GarbageCollected](#) &gc)
Add friendly output.

5.8.1 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.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 [GarbageCollected\(\)](#) [1/2]

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

Copy Constructor.

Parameters

The	other GarbageCollected object to copy.
-----	--

5.8.2.2 GarbageCollected() [2/2]

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

Move Constructor.

Parameters

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

5.8.2.3 ~GarbageCollected()

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

Destructor.

Clean up the tracked object, if appropriate.

5.8.3 Member Function Documentation

5.8.3.1 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:

**5.8.3.2 operator*()**

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

Access the tracked object.

Returns

A reference to the tracked object.

5.8.3.3 operator->()

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

Access the tracked object as a pointer.

Returns

A pointer to the tracked object.

5.8.3.4 operator=() [1/2]

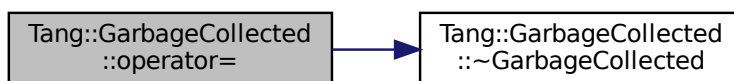
```
GarbageCollected& Tang::GarbageCollected::operator= (
    const GarbageCollected & other ) [inline]
```

Copy Assignment.

Parameters

<i>The</i>	other GarbageCollected object.
------------	--

Here is the call graph for this function:



5.8.3.5 operator=() [2/2]

```

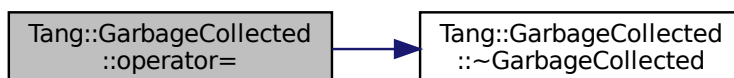
GarbageCollected& Tang::GarbageCollected::operator= (
    GarbageCollected && other ) [inline]
  
```

Move Assignment.

Parameters

<i>The</i>	other GarbageCollected object.
------------	--

Here is the call graph for this function:



5.8.3.6 operator==(

```

bool Tang::GarbageCollected::operator== (
    auto & val ) const [inline]
  
```

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

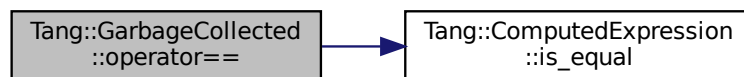
Parameters

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

Returns

True if they are equal, false otherwise.

Here is the call graph for this function:



5.8.4 Friends And Related Function Documentation

5.8.4.1 operator<<

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

Add friendly output.

Parameters

<i>out</i>	The output stream.
<i>gc</i>	The GarbageCollected value.

Returns

The output stream.

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

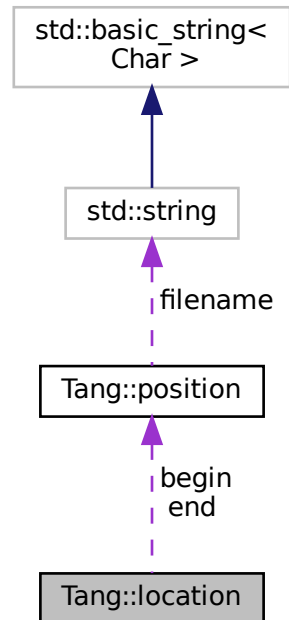
- [include/garbageCollected.hpp](#)

5.9 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.

5.9.1 Detailed Description

Two points in a source file.

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

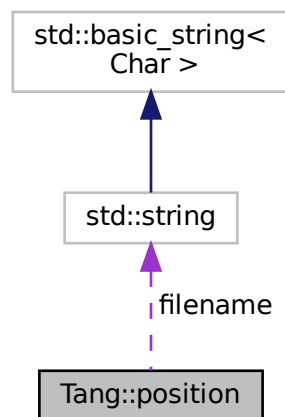
- [build/generated/location.hh](#)

5.10 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.

5.10.1 Detailed Description

A point in a source file.

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

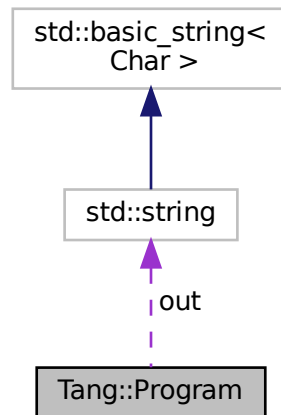
- `build/generated/location.hh`

5.11 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 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.

5.11.1 Detailed Description

Represents a compiled script or template that may be executed.

5.11.2 Member Enumeration Documentation

5.11.2.1 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.

5.11.3 Constructor & Destructor Documentation

5.11.3.1 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> .

5.11.4 Member Function Documentation

5.11.4.1 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.
-----------	-----------------------------------

5.11.4.2 dumpBytecode()

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

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

Returns

A string containing the Opcode representation.

5.11.4.3 execute()

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

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

Returns

The current [Program](#) object.

5.11.4.4 getAst()

```
optional< const 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.

5.11.4.5 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.

5.11.4.6 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.cpp](#)

5.12 Tang::SingletonObjectPool< T > Class Template Reference

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.

5.12.1 Member Function Documentation

5.12.1.1 `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.

5.12.1.2 `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.

5.12.1.3 `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](#)

5.13 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](#).

5.13.1 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).

5.13.2 Constructor & Destructor Documentation

5.13.2.1 TangBase()

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

The constructor.

Isn't it glorious.

5.13.3 Member Function Documentation

5.13.3.1 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:

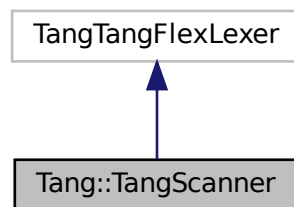
- [include/tangBase.hpp](#)
- [src/tangBase.cpp](#)

5.14 Tang::TangScanner Class Reference

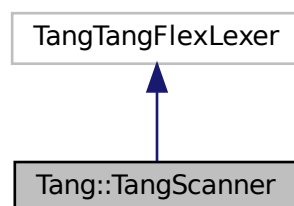
The Flex lexer class for the main Tang language.

```
#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.

5.14.1 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.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 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)

5.14.3 Member Function Documentation

5.14.3.1 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

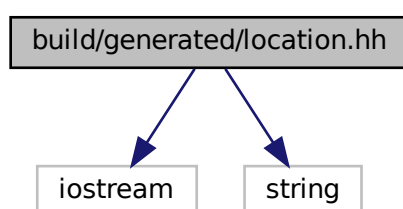
6.1 build/generated/location.hh File Reference

Define the Tang ::location class.

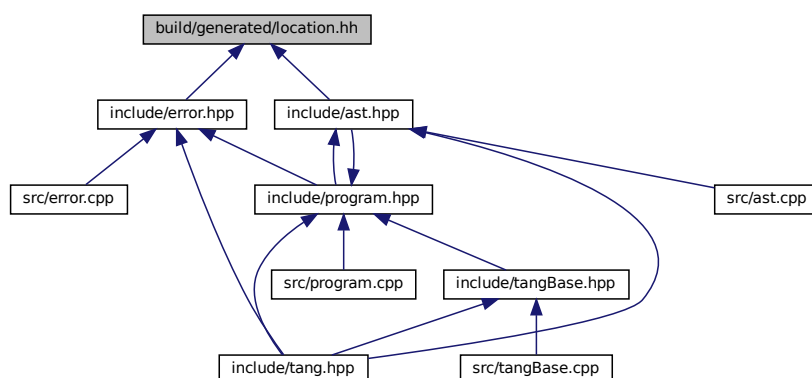
```
#include <iostream>
```

```
#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<<() [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/ast.hpp File Reference

Define the [Tang::AstNode](#) and its associated/derivative classes.

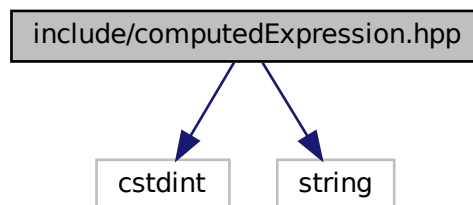
```
#include <string>
#include "location.hh"
#include "program.hpp"
```


6.3 include/computedExpression.hpp File Reference

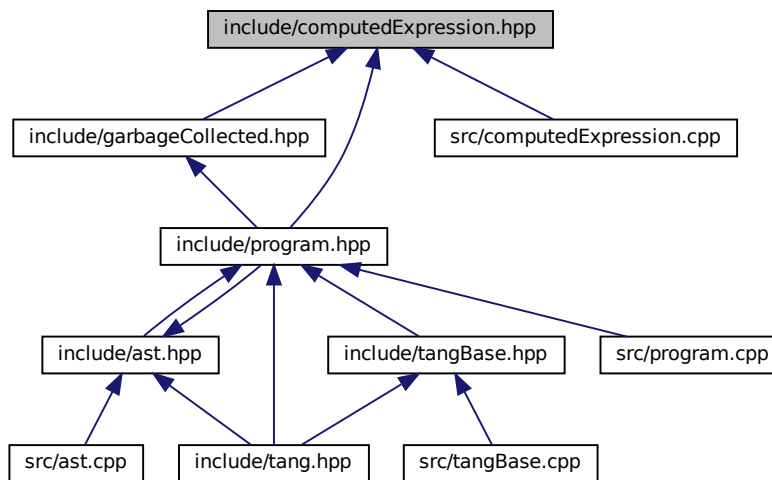
```
#include <cstdint>
```

```
#include <string>
```

Include dependency graph for computedExpression.hpp:



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



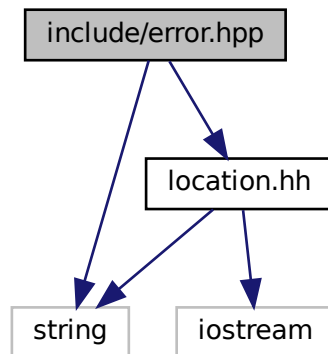
Classes

- class [Tang::ComputedExpression](#)
Represents the result of a computation that has been executed.
- class [Tang::ComputedExpressionInteger](#)
Represents an Integer that is the result of a computation.
- class [Tang::ComputedExpressionFloat](#)
Represents a Float that is the result of a computation.

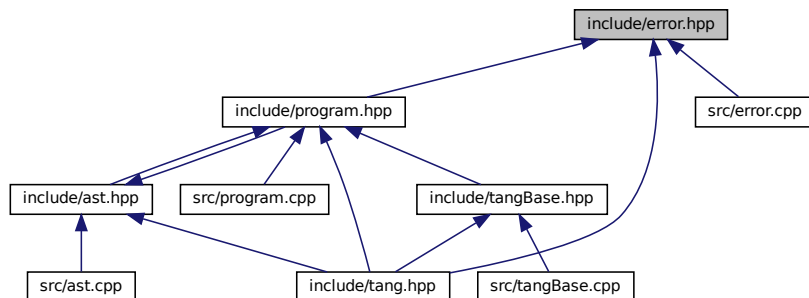
6.4 include/error.hpp File Reference

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

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



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



Classes

- class [Tang::Error](#)

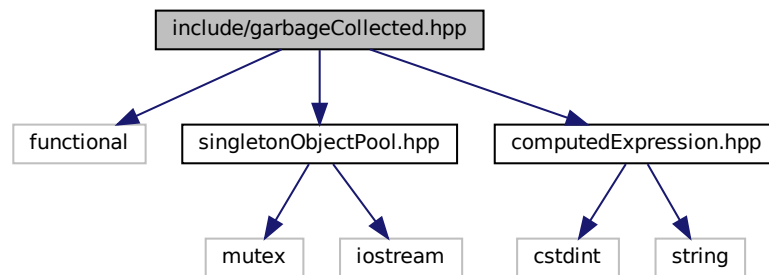
The [Error](#) class is used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error.

6.4.1 Detailed Description

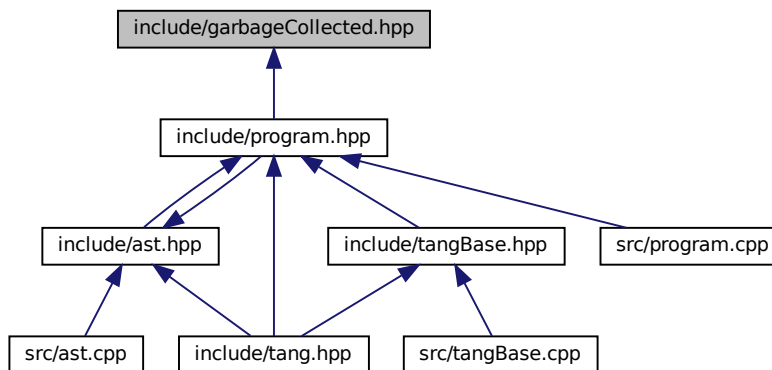
Define the [Tang::Error](#) class used to describe syntax and runtime errors.

6.5 include/garbageCollected.hpp File Reference

```
#include <functional>
#include "singletonObjectPool.hpp"
#include "computedExpression.hpp"
Include dependency graph for garbageCollected.hpp:
```



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



Classes

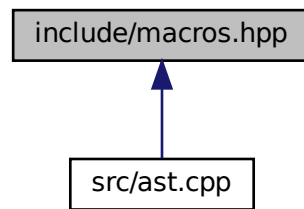
- class [Tang::GarbageCollected](#)

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

6.6 include/macros.hpp File Reference

Contains generic macros.

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



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.

6.6.1 Detailed Description

Contains generic macros.

6.6.2 Macro Definition Documentation

6.6.2.1 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)) {}`

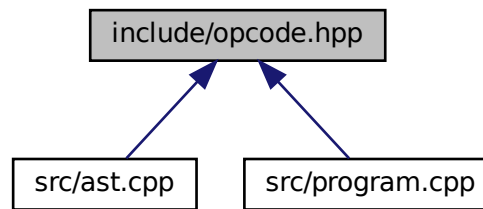
Parameters

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

6.7 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](#) { [INTEGER](#) , [FLOAT](#) }

6.7.1 Detailed Description

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

6.7.2 Enumeration Type Documentation

6.7.2.1 Opcode

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

Enumerator

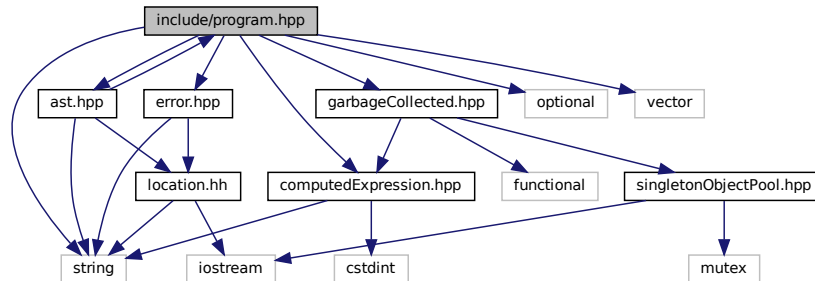
INTEGER	Push an integer onto the stack.
FLOAT	Push a floating point number onto the stack.

6.8 include/program.hpp File Reference

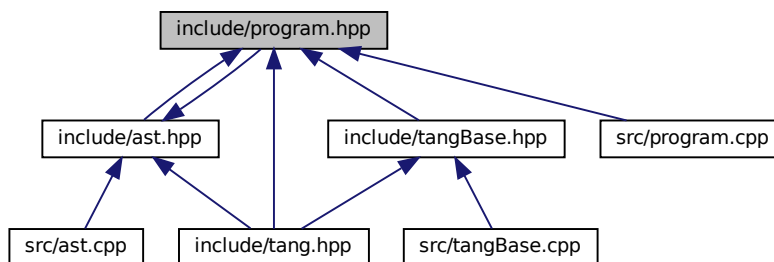
Define the [Tang::Program](#) class used to compile and execute source code.

```
#include <string>
#include <optional>
#include <vector>
#include "ast.hpp"
```

```
#include "error.hpp"
#include "computedExpression.hpp"
#include "garbageCollected.hpp"
Include dependency graph for program.hpp:
```



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



Classes

- class [Tang::Program](#)
Represents a compiled script or template that may be executed.

Typedefs

- using [Tang::Bytecode](#) = `std::vector< uint64_t >`
Contains the Opcodes of a compiled program.

6.8.1 Detailed Description

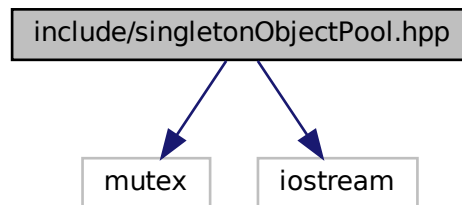
Define the [Tang::Program](#) class used to compile and execute source code.

6.9 include/singletonObjectPool.hpp File Reference

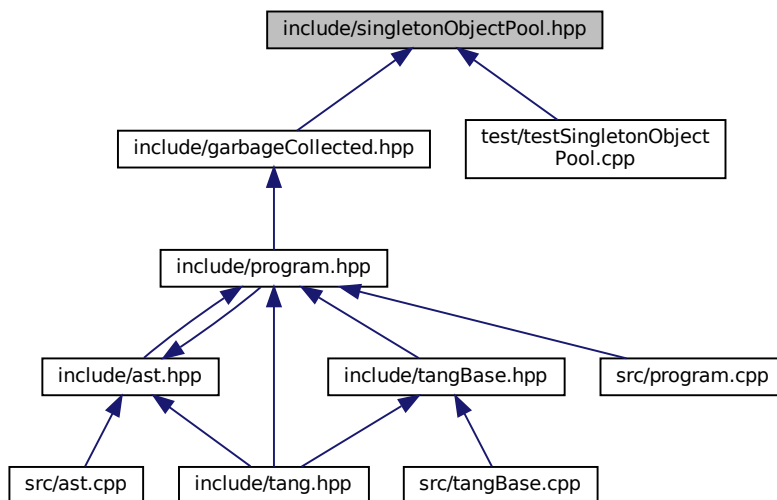
```
#include <mutex>
```

```
#include <iostream>
```

Include dependency graph for singletonObjectPool.hpp:



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



Classes

- class [Tang::SingletonObjectPool< T >](#)

Macros

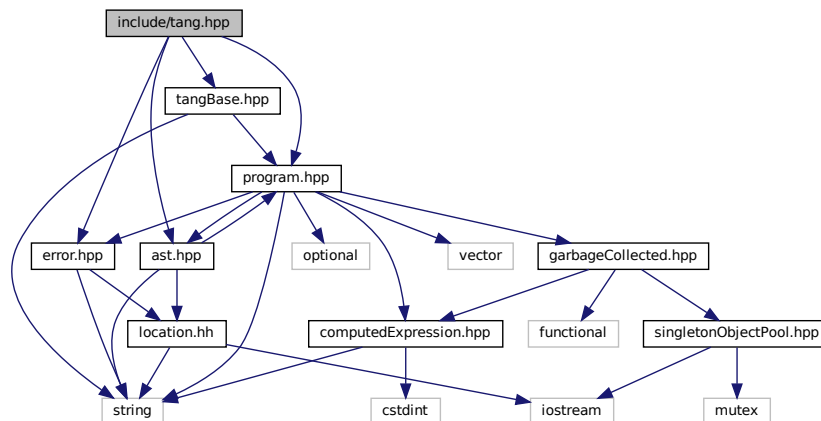
- #define [GROW](#) 1024

The threshold size to use when allocating blocks of data, measured in the number of instances of the object type.

6.10 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 "ast.hpp"
#include "error.hpp"
#include "program.hpp"
Include dependency graph for tang.hpp:
```



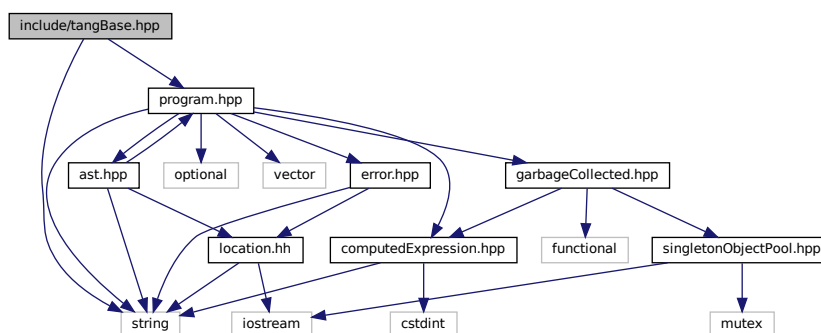
6.10.1 Detailed Description

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

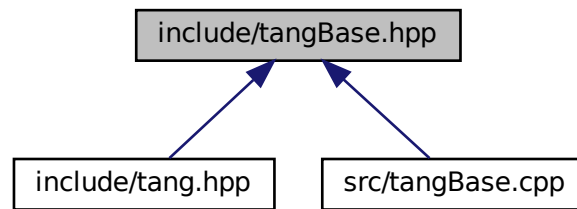
6.11 include/tangBase.hpp File Reference

Defines 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.

6.11.1 Detailed Description

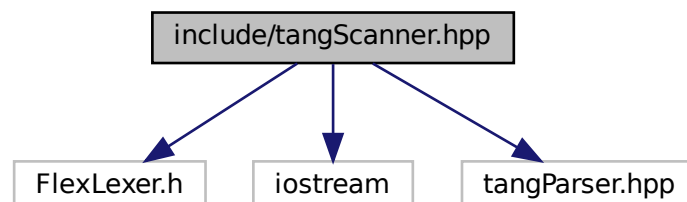
Defines the [Tang::TangBase](#) class used to interact with Tang.

6.12 include/tangScanner.hpp File Reference

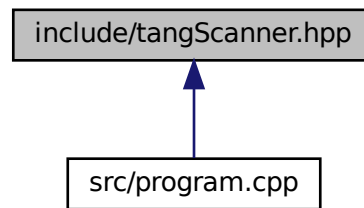
Defines the [Tang::TangScanner](#) used to tokenize a Tang script.

```
#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

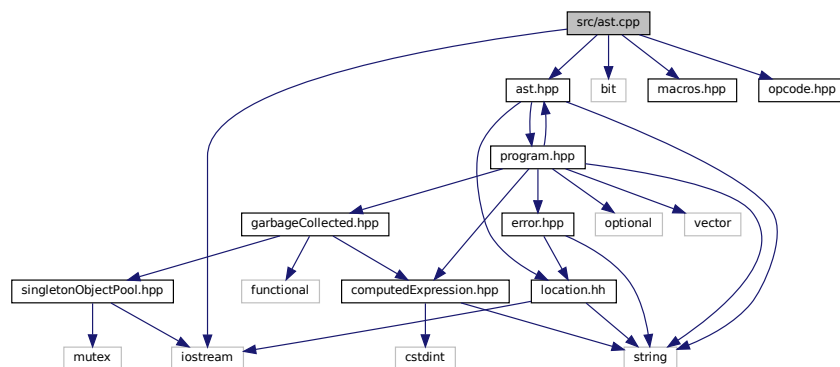
- `#define yyFlexLexer TangTangFlexLexer`
- `#define YY_DECL Tang::TangParser::symbol_type Tang::TangScanner::get_next_token\(\)`

6.12.1 Detailed Description

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

6.13 src/ast.cpp File Reference

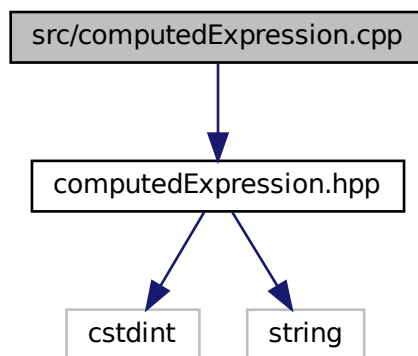
```
#include <iostream>
#include <bit>
#include "ast.hpp"
#include "macros.hpp"
#include "opcode.hpp"
Include dependency graph for ast.cpp:
```



6.14 src/computedExpression.cpp File Reference

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

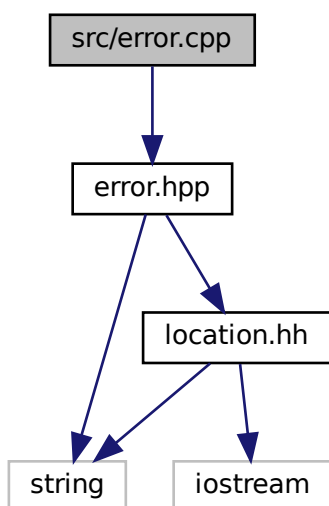
Include dependency graph for computedExpression.cpp:



6.15 src/error.cpp File Reference

```
#include "error.hpp"
```

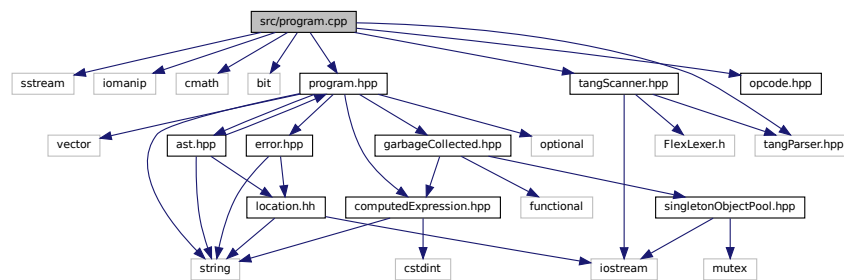
Include dependency graph for error.cpp:



6.16 src/program.cpp File Reference

```
#include <sstream>
#include <iomanip>
#include <cmath>
#include <bit>
#include "program.hpp"
#include "tangScanner.hpp"
#include "tangParser.hpp"
#include "opcode.hpp"
```

Include dependency graph for program.cpp:



Macros

- `#define DUMPPROGRAMCHECK(x)`
Verify the size of the Bytecode vector so that it may be safely accessed.
- `#define EXECUTEPROGRAMCHECK(x)`
Verify the size of the Bytecode vector so that it may be safely accessed.

6.16.1 Macro Definition Documentation

6.16.1.1 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.
---	--

6.16.1.2 EXECUTEPROGRAMCHECK

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

Value:

```
if (this->bytecode.size() < (pc + (x))) { \
    /* TODO push an error on to the stack! */ \
    pc = this->bytecode.size(); \
    break; \
}
```

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

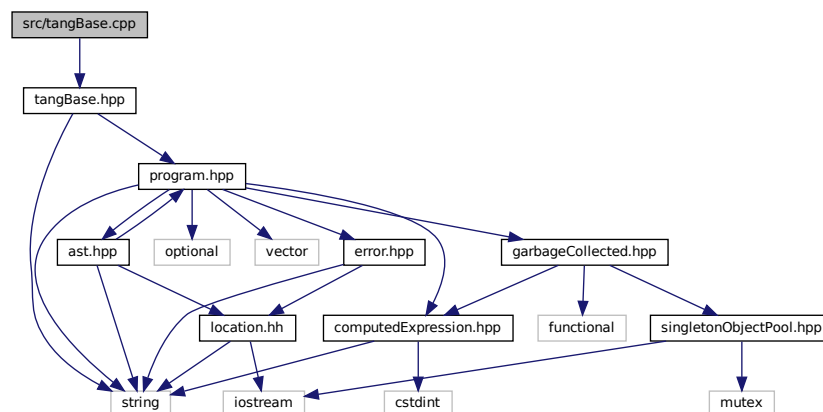
Parameters

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

6.17 src/tangBase.cpp File Reference

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

Include dependency graph for tangBase.cpp:

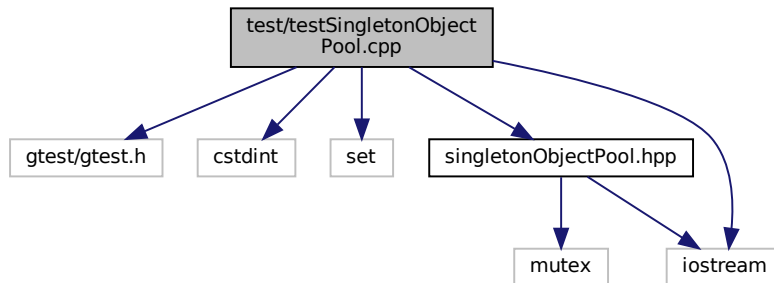


6.18 test/testSingletonObjectPool.cpp File Reference

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

Index

- ~GarbageCollected
 - Tang::GarbageCollected, [29](#)
- addBytecode
 - Tang::Program, [37](#)
- AstNode
 - Tang::AstNode, [11](#)
- AstNodeFloat
 - Tang::AstNodeFloat, [14](#)
- AstNodeInteger
 - Tang::AstNodeInteger, [17](#)
- build/generated/location.hh, [45](#)
- CodeType
 - Tang::Program, [37](#)
- compileScript
 - Tang::TangBase, [41](#)
- ComputedExpressionFloat
 - Tang::ComputedExpressionFloat, [21](#)
- ComputedExpressionInteger
 - Tang::ComputedExpressionInteger, [24](#)
- dump
 - Tang::ComputedExpression, [18](#)
 - Tang::ComputedExpressionFloat, [21](#)
 - Tang::ComputedExpressionInteger, [24](#)
- dumpBytecode
 - Tang::Program, [38](#)
- DUMPPROGRAMCHECK
 - program.cpp, [60](#)
- Error
 - Tang::Error, [27](#)
- execute
 - Tang::Program, [38](#)
- EXECUTEPROGRAMCHECK
 - program.cpp, [61](#)
- FLOAT
 - opcode.hpp, [53](#)
- GarbageCollected
 - Tang::GarbageCollected, [28](#), [29](#)
- get
 - Tang::SingletonObjectPool< T >, [39](#)
- get_next_token
 - Tang::TangScanner, [43](#)
- getAst
 - Tang::Program, [38](#)
- getCode
 - Tang::Program, [38](#)
- getInstance
 - Tang::SingletonObjectPool< T >, [40](#)
- getResult
 - Tang::Program, [39](#)
- include/ast.hpp, [47](#)
- include/computedExpression.hpp, [49](#)
- include/error.hpp, [50](#)
- include/garbageCollected.hpp, [51](#)
- include/macros.hpp, [51](#)
- include/opcode.hpp, [52](#)
- include/program.hpp, [53](#)
- include/singletonObjectPool.hpp, [55](#)
- include/tang.hpp, [56](#)
- include/tangBase.hpp, [56](#)
- include/tangScanner.hpp, [57](#)
- INTEGER
 - opcode.hpp, [53](#)
- is_equal
 - Tang::ComputedExpression, [18](#), [19](#)
 - Tang::ComputedExpressionFloat, [21](#), [22](#)
 - Tang::ComputedExpressionInteger, [24](#), [25](#)
- location.hh
 - operator<<, [46](#), [47](#)
- macros.hpp
 - TANG_UNUSED, [52](#)
- make
 - Tang::GarbageCollected, [29](#)
- makeCopy
 - Tang::AstNode, [11](#)
 - Tang::AstNodeFloat, [14](#)
 - Tang::AstNodeInteger, [17](#)
 - Tang::ComputedExpression, [19](#)
 - Tang::ComputedExpressionFloat, [22](#)
 - Tang::ComputedExpressionInteger, [25](#)
- Opcode
 - opcode.hpp, [53](#)
- opcode.hpp
 - FLOAT, [53](#)
 - INTEGER, [53](#)
 - Opcode, [53](#)
- operator<<
 - location.hh, [46](#), [47](#)
 - Tang::GarbageCollected, [32](#)
- operator*
 - Tang::GarbageCollected, [30](#)

- operator->
 - Tang::GarbageCollected, 30
- operator=
 - Tang::GarbageCollected, 30, 31
- operator==
 - Tang::GarbageCollected, 31
- Program
 - Tang::Program, 37
- program.cpp
 - DUMPPROGRAMCHECK, 60
 - EXECUTEPROGRAMCHECK, 61
- recycle
 - Tang::SingletonObjectPool< T >, 40
- Script
 - Tang::Program, 37
- src/ast.cpp, 58
- src/computedExpression.cpp, 59
- src/error.cpp, 59
- src/program.cpp, 60
- src/tangBase.cpp, 61
- Tang::AstNode, 9
 - AstNode, 11
 - makeCopy, 11
- Tang::AstNodeFloat, 12
 - AstNodeFloat, 14
 - makeCopy, 14
- Tang::AstNodeInteger, 15
 - AstNodeInteger, 17
 - makeCopy, 17
- Tang::ComputedExpression, 18
 - dump, 18
 - is_equal, 18, 19
 - makeCopy, 19
- Tang::ComputedExpressionFloat, 20
 - ComputedExpressionFloat, 21
 - dump, 21
 - is_equal, 21, 22
 - makeCopy, 22
- Tang::ComputedExpressionInteger, 23
 - ComputedExpressionInteger, 24
 - dump, 24
 - is_equal, 24, 25
 - makeCopy, 25
- Tang::Error, 26
 - Error, 27
- Tang::GarbageCollected, 27
 - ~GarbageCollected, 29
 - GarbageCollected, 28, 29
 - make, 29
 - operator<<, 32
 - operator*, 30
 - operator->, 30
 - operator=, 30, 31
 - operator==, 31
- Tang::location, 32
- Tang::position, 34
- Tang::Program, 35
 - addBytecode, 37
 - CodeType, 37
 - dumpBytecode, 38
 - execute, 38
 - getAst, 38
 - getCode, 38
 - getResult, 39
 - Program, 37
 - Script, 37
 - Template, 37
- Tang::SingletonObjectPool< T >, 39
 - get, 39
 - getInstance, 40
 - recycle, 40
- Tang::TangBase, 40
 - compileScript, 41
 - TangBase, 41
- Tang::TangScanner, 42
 - get_next_token, 43
 - TangScanner, 43
- TANG_UNUSED
 - macros.hpp, 52
- TangBase
 - Tang::TangBase, 41
- TangScanner
 - Tang::TangScanner, 43
- Template
 - Tang::Program, 37
- test/testSingletonObjectPool.cpp, 61