

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::AstNodeInteger Class Reference	12
5.2.1 Detailed Description	14
5.2.2 Constructor & Destructor Documentation	14
5.2.2.1 AstNodeInteger()	14
5.2.3 Member Function Documentation	14
5.2.3.1 makeCopy()	14
5.3 Tang::ComputedExpression Class Reference	15
5.3.1 Detailed Description	15
5.3.2 Member Function Documentation	15
5.3.2.1 dump()	15
5.3.2.2 makeCopy()	16
5.4 Tang::ComputedExpressionInteger Class Reference	16
5.4.1 Detailed Description	17
5.4.2 Constructor & Destructor Documentation	17
5.4.2.1 ComputedExpressionInteger()	17
5.4.3 Member Function Documentation	17
5.4.3.1 dump()	17
5.4.3.2 makeCopy()	18
5.5 Tang::Error Class Reference	18
5.5.1 Detailed Description	19
5.5.2 Constructor & Destructor Documentation	20
5.5.2.1 Error()	20
5.6 Tang::location Class Reference	20

5.6.1 Detailed Description . . . . .	22
5.7 Tang::position Class Reference . . . . .	22
5.7.1 Detailed Description . . . . .	23
5.8 Tang::Program Class Reference . . . . .	23
5.8.1 Detailed Description . . . . .	25
5.8.2 Member Enumeration Documentation . . . . .	25
5.8.2.1 CodeType . . . . .	25
5.8.3 Constructor & Destructor Documentation . . . . .	25
5.8.3.1 Program() . . . . .	25
5.8.4 Member Function Documentation . . . . .	25
5.8.4.1 addBytecode() . . . . .	26
5.8.4.2 dumpBytecode() . . . . .	26
5.8.4.3 execute() . . . . .	26
5.8.4.4 getAst() . . . . .	26
5.8.4.5 getCode() . . . . .	27
5.8.4.6 getResult() . . . . .	27
5.9 Tang::TangBase Class Reference . . . . .	27
5.9.1 Detailed Description . . . . .	27
5.9.2 Constructor & Destructor Documentation . . . . .	28
5.9.2.1 TangBase() . . . . .	28
5.9.3 Member Function Documentation . . . . .	28
5.9.3.1 compileScript() . . . . .	28
5.10 Tang::TangScanner Class Reference . . . . .	28
5.10.1 Detailed Description . . . . .	29
5.10.2 Constructor & Destructor Documentation . . . . .	29
5.10.2.1 TangScanner() . . . . .	30
5.10.3 Member Function Documentation . . . . .	30
5.10.3.1 get_next_token() . . . . .	30
<b>6 File Documentation</b> . . . . .	<b>31</b>
6.1 build/generated/location.hh File Reference . . . . .	31
6.1.1 Detailed Description . . . . .	32
6.1.2 Function Documentation . . . . .	32
6.1.2.1 operator<<() [1/2] . . . . .	33
6.1.2.2 operator<<() [2/2] . . . . .	33
6.2 include/ast.hpp File Reference . . . . .	33
6.2.1 Detailed Description . . . . .	34
6.3 include/computedExpression.hpp File Reference . . . . .	35
6.4 include/error.hpp File Reference . . . . .	35
6.4.1 Detailed Description . . . . .	36
6.5 include/macros.hpp File Reference . . . . .	37
6.5.1 Detailed Description . . . . .	37

6.5.2 Macro Definition Documentation . . . . .	37
6.5.2.1 TANG_UNUSED . . . . .	37
6.6 include/opcode.hpp File Reference . . . . .	38
6.6.1 Detailed Description . . . . .	38
6.6.2 Enumeration Type Documentation . . . . .	38
6.6.2.1 Opcode . . . . .	38
6.7 include/program.hpp File Reference . . . . .	38
6.7.1 Detailed Description . . . . .	39
6.8 include/tang.hpp File Reference . . . . .	40
6.8.1 Detailed Description . . . . .	40
6.9 include/tangBase.hpp File Reference . . . . .	40
6.9.1 Detailed Description . . . . .	41
6.10 include/tangScanner.hpp File Reference . . . . .	41
6.10.1 Detailed Description . . . . .	42
6.11 src/ast.cpp File Reference . . . . .	43
6.12 src/computedExpression.cpp File Reference . . . . .	43
6.13 src/error.cpp File Reference . . . . .	44
6.14 src/program.cpp File Reference . . . . .	44
6.14.1 Macro Definition Documentation . . . . .	45
6.14.1.1 DUMPPROGRAMCHECK . . . . .	45
6.14.1.2 EXECUTEPROGRAMCHECK . . . . .	45
6.15 src/tangBase.cpp File Reference . . . . .	46
<b>Index</b>	<b>47</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::AstNodeInteger . . . . .	12
Tang::ComputedExpression . . . . .	15
Tang::ComputedExpressionInteger . . . . .	16
Tang::Error . . . . .	18
Tang::location . . . . .	20
Tang::position . . . . .	22
Tang::Program . . . . .	23
Tang::TangBase . . . . .	27
TangTangFlexLexer	
Tang::TangScanner . . . . .	28



## 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::AstNodeInteger</a>	An <a href="#">AstNode</a> that represents an integer literal . . . . .	12
<a href="#">Tang::ComputedExpression</a>	Represents the result of a computation that has been executed . . . . .	15
<a href="#">Tang::ComputedExpressionInteger</a>	Represents an Integer that is the result of a computation . . . . .	16
<a href="#">Tang::Error</a>	Used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error . . . . .	18
<a href="#">Tang::location</a>	Two points in a source file . . . . .	20
<a href="#">Tang::position</a>	A point in a source file . . . . .	22
<a href="#">Tang::Program</a>	Represents a compiled script or template that may be executed . . . . .	23
<a href="#">Tang::TangBase</a>	The base class for the Tang programming language . . . . .	27
<a href="#">Tang::TangScanner</a>	The Flex lexer class for the main Tang language . . . . .	28



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all documented files with brief descriptions:

build/generated/ <a href="#">location.hh</a>	
Define the <code>Tang::location</code> class . . . . .	31
include/ <a href="#">ast.hpp</a>	
Define the <code>Tang::AstNode</code> and its associated/derivative classes . . . . .	33
include/ <a href="#">computedExpression.hpp</a>	35
include/ <a href="#">error.hpp</a>	
Define the <code>Tang::Error</code> class used to describe syntax and runtime errors . . . . .	35
include/ <a href="#">macros.hpp</a>	
Contains generic macros . . . . .	37
include/ <a href="#">opcode.hpp</a>	
Declare the Opcodes used in the Bytecode representation of a program . . . . .	38
include/ <a href="#">program.hpp</a>	
Define the <code>Tang::Program</code> class used to compile and execute source code . . . . .	38
include/ <a href="#">tang.hpp</a>	
Header file supplied for use by 3rd party code so that they can easily include all necessary headers . . . . .	40
include/ <a href="#">tangBase.hpp</a>	
Defines the <code>Tang::TangBase</code> class used to interact with Tang . . . . .	40
include/ <a href="#">tangScanner.hpp</a>	
Defines the <code>Tang::TangScanner</code> used to tokenize a Tang script . . . . .	41
src/ <a href="#">ast.cpp</a>	43
src/ <a href="#">computedExpression.cpp</a>	43
src/ <a href="#">error.cpp</a>	44
src/ <a href="#">program.cpp</a>	44
src/ <a href="#">tangBase.cpp</a>	46



## Chapter 5

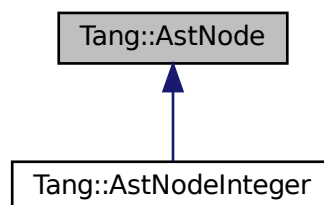
# Class Documentation

### 5.1 Tang::AstNode Class Reference

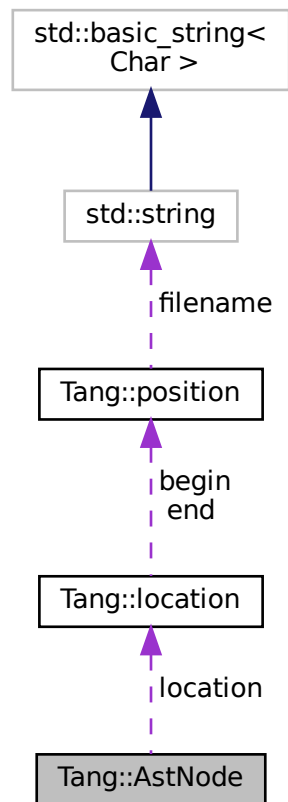
Base class for representing nodes of an Abstract Syntax Tree (AST).

```
#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::AstNodeInteger](#).

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

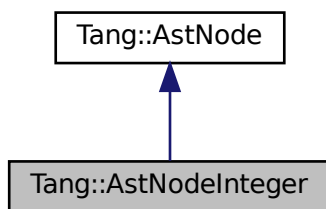
- [include/ast.hpp](#)
- [src/ast.cpp](#)

## 5.2 Tang::AstNodeInteger Class Reference

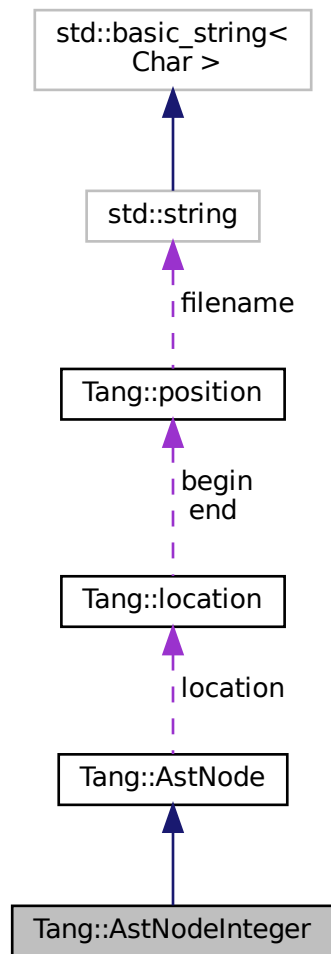
An [AstNode](#) that represents an integer literal.

```
#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.*

### 5.2.1 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.

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 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.

### 5.2.3 Member Function Documentation

#### 5.2.3.1 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:

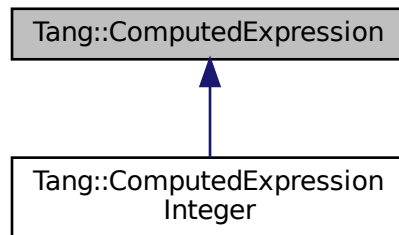
- include/[ast.hpp](#)
- src/[ast.cpp](#)

## 5.3 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).*

#### 5.3.1 Detailed Description

Represents the result of a computation that has been executed.

#### 5.3.2 Member Function Documentation

##### 5.3.2.1 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::ComputedExpressionInteger](#).

### 5.3.2.2 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](#).

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

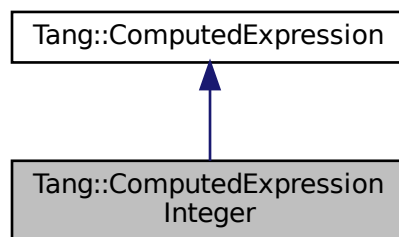
- [include/computedExpression.hpp](#)
- [src/computedExpression.cpp](#)

## 5.4 Tang::ComputedExpressionInteger Class Reference

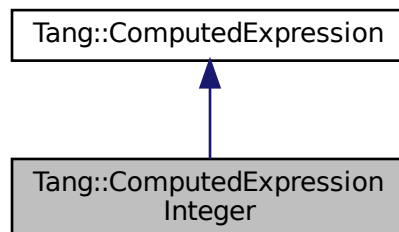
Represents an Integer that is the result of a computation.

```
#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).*

### 5.4.1 Detailed Description

Represents an Integer that is the result of a computation.

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 ComputedExpressionInteger()

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

Construct an Integer result.

##### Parameters

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

### 5.4.3 Member Function Documentation

#### 5.4.3.1 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](#).

### 5.4.3.2 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](#)

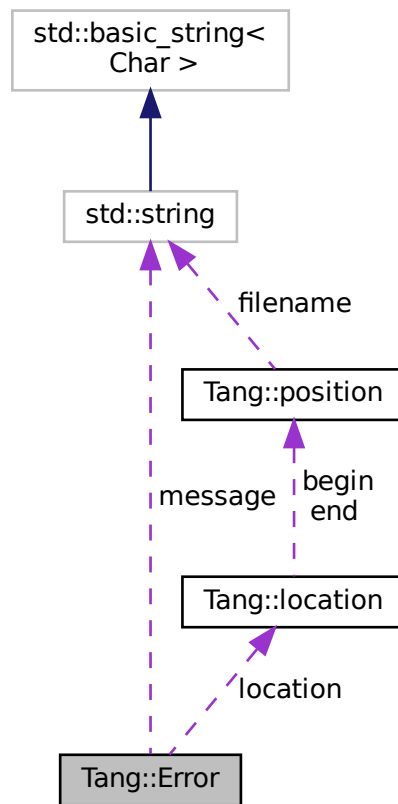
## 5.5 Tang::Error Class Reference

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.*

### 5.5.1 Detailed Description

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

## 5.5.2 Constructor & Destructor Documentation

### 5.5.2.1 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:

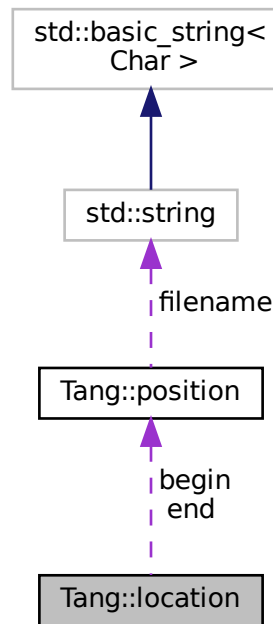
- [include/error.hpp](#)
- [src/error.cpp](#)

## 5.6 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.*

### 5.6.1 Detailed Description

Two points in a source file.

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

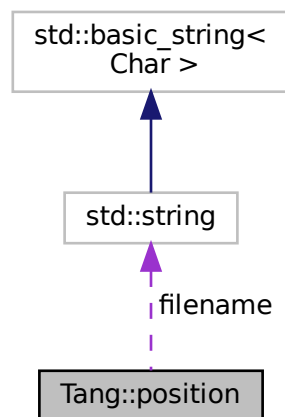
- [build/generated/location.hh](#)

## 5.7 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.*

### 5.7.1 Detailed Description

A point in a source file.

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

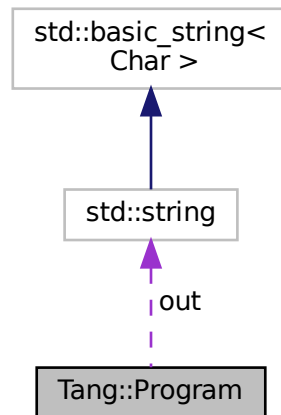
- `build/generated/location.hh`

## 5.8 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 ComputedExpression * >` `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.*

### 5.8.1 Detailed Description

Represents a compiled script or template that may be executed.

### 5.8.2 Member Enumeration Documentation

#### 5.8.2.1 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.

### 5.8.3 Constructor & Destructor Documentation

#### 5.8.3.1 Program()

```
Program::Program (
    std::string code,
    Program::CodeType codeType )
```

Create a compiled program using the provided code.

##### Parameters

<i>code</i>	The code to be compiled.
<i>codeType</i>	Whether the code is a <code>Script</code> or <code>Template</code> .

### 5.8.4 Member Function Documentation

#### 5.8.4.1 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.
-----------	-----------------------------------

#### 5.8.4.2 dumpBytecode()

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

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

##### Returns

A string containing the Opcode representation.

#### 5.8.4.3 execute()

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

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

##### Returns

The current [Program](#) object.

#### 5.8.4.4 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.



#### 5.8.4.5 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.

#### 5.8.4.6 getResult()

```
optional< const ComputedExpression * > 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](#)

## 5.9 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](#).*

### 5.9.1 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).

## 5.9.2 Constructor & Destructor Documentation

### 5.9.2.1 TangBase()

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

The constructor.

Isn't it glorious.

## 5.9.3 Member Function Documentation

### 5.9.3.1 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:

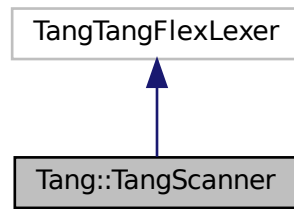
- include/[tangBase.hpp](#)
- src/[tangBase.cpp](#)

## 5.10 Tang::TangScanner Class Reference

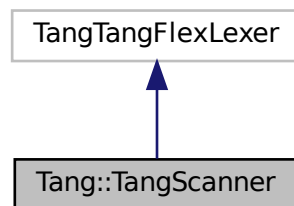
The Flex lexer class for the main Tang language.

```
#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.*

### 5.10.1 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.

### 5.10.2 Constructor & Destructor Documentation

### 5.10.2.1 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)

## 5.10.3 Member Function Documentation

### 5.10.3.1 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

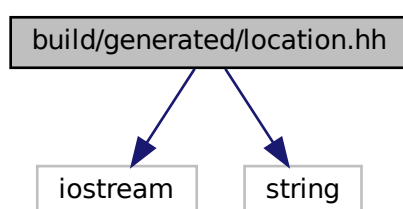
### 6.1 build/generated/location.hh File Reference

Define the Tang ::location class.

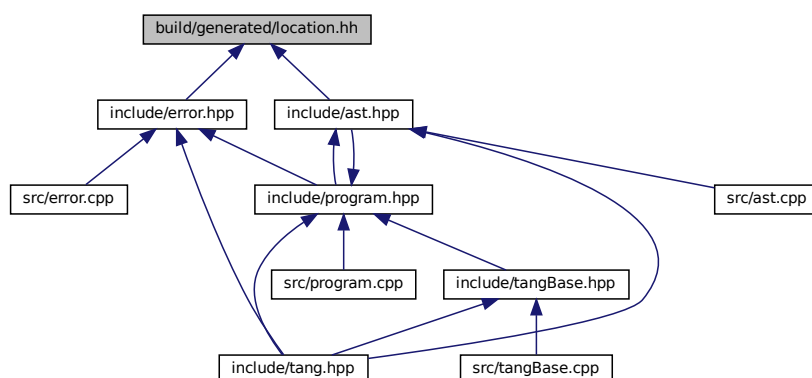
```
#include <iostream>
```

```
#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**

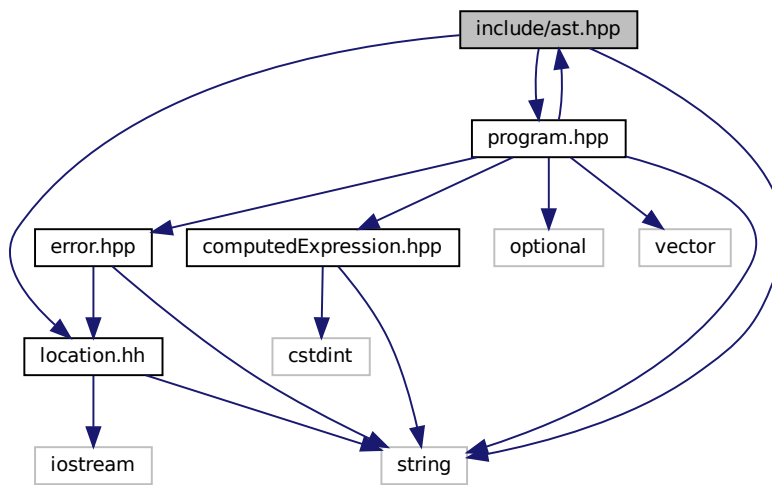
<i>ostr</i>	the destination output stream
<i>pos</i>	a reference to the position to redirect

**6.2 include/ast.hpp File Reference**

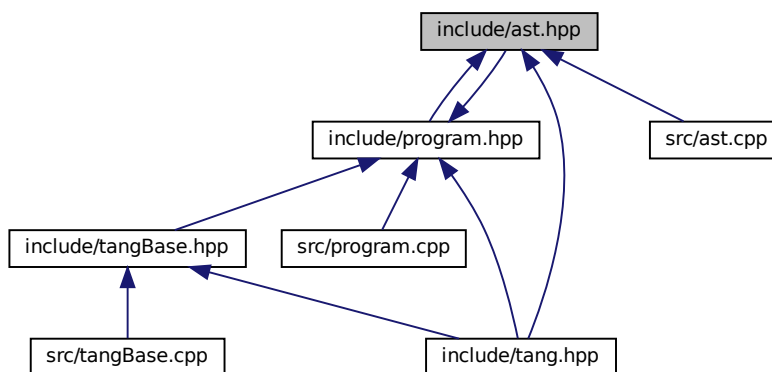
Define the [Tang::AstNode](#) and its associated/derivative classes.

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

Include dependency graph for ast.hpp:



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



## Classes

- class [Tang::AstNode](#)  
Base class for representing nodes of an Abstract Syntax Tree (AST).
- class [Tang::AstNodeInteger](#)  
An [AstNode](#) that represents an integer literal.

### 6.2.1 Detailed Description

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

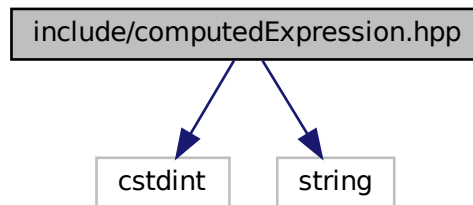


## 6.3 include/computedExpression.hpp File Reference

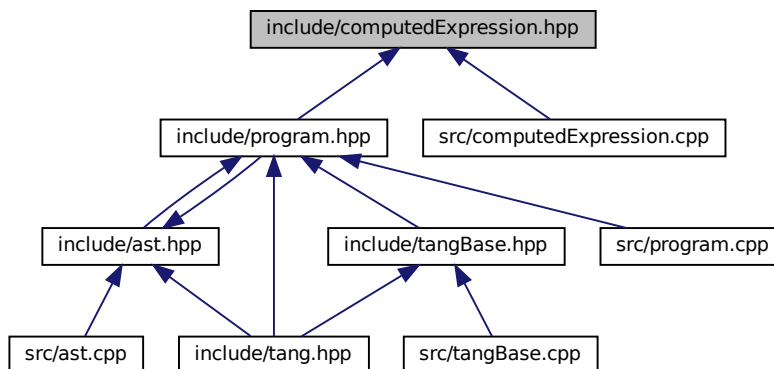
```
#include <stdint>
```

```
#include <string>
```

Include dependency graph for computedExpression.hpp:



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



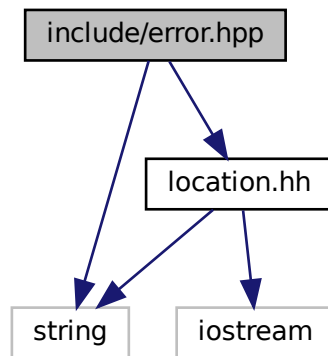
### Classes

- 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.*

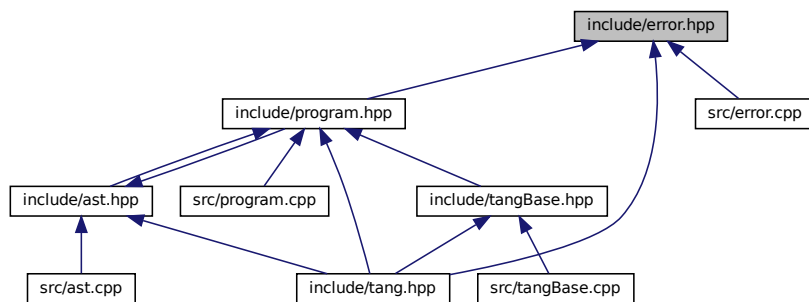
## 6.4 include/error.hpp File Reference

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.

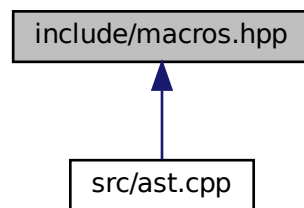
### 6.4.1 Detailed Description

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

## 6.5 include/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.*

### 6.5.1 Detailed Description

Contains generic macros.

### 6.5.2 Macro Definition Documentation

#### 6.5.2.1 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)) {}`

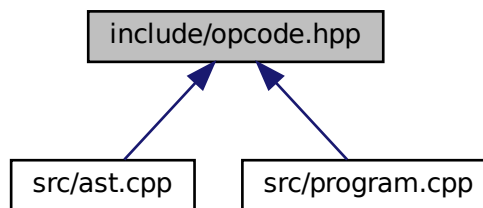
#### Parameters

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

## 6.6 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](#) }

#### 6.6.1 Detailed Description

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

#### 6.6.2 Enumeration Type Documentation

##### 6.6.2.1 Opcode

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

Enumerator

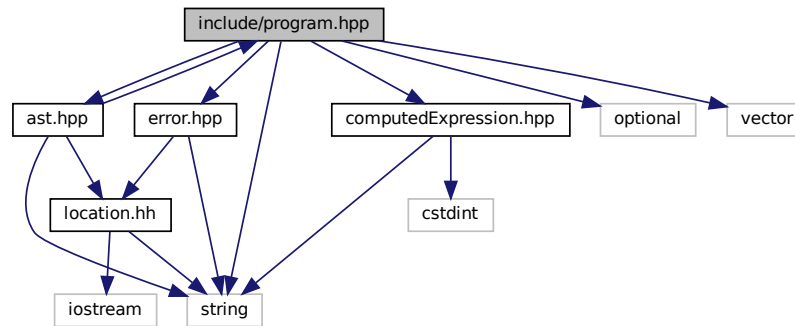
INTEGER	Push an integer onto the stack.
---------	---------------------------------

## 6.7 include/program.hpp File Reference

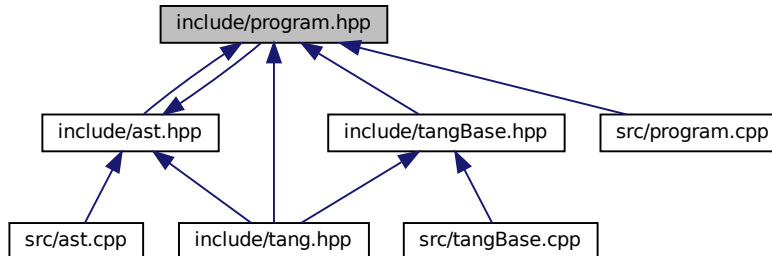
Define the [Tang::Program](#) class used to compile and execute source code.

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

Include 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.*

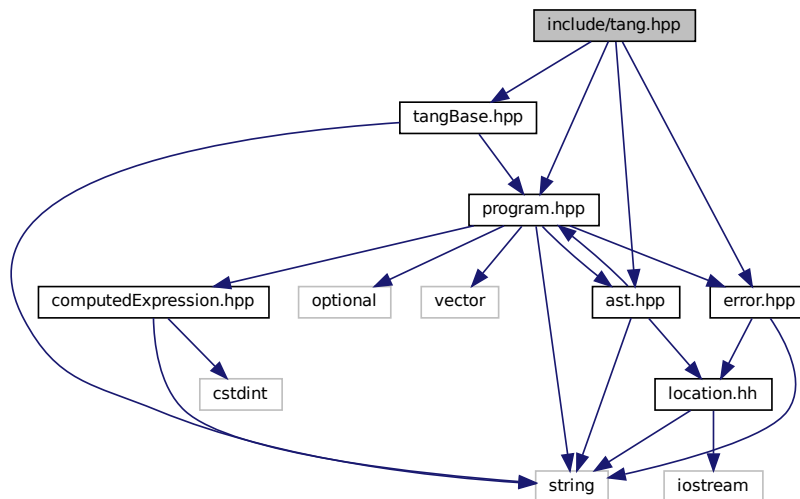
### 6.7.1 Detailed Description

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

## 6.8 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:
```



### 6.8.1 Detailed Description

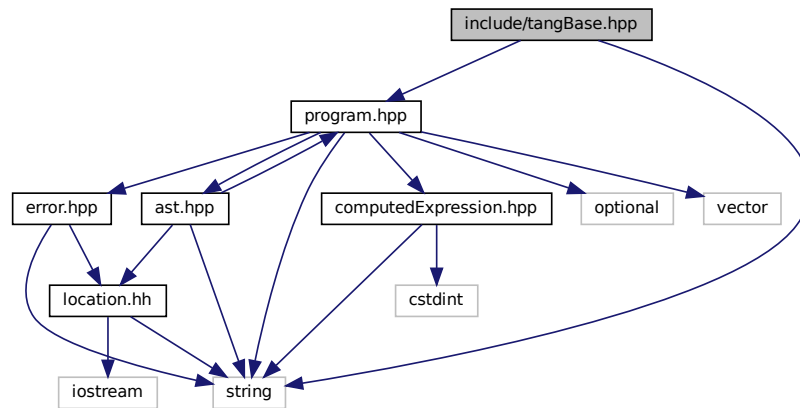
Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

## 6.9 include/tangBase.hpp File Reference

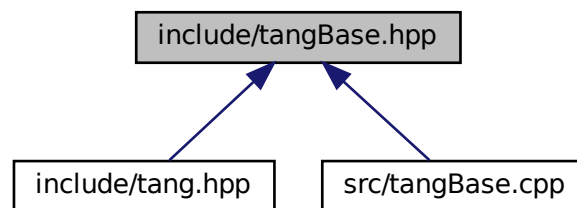
Defines the [Tang::TangBase](#) class used to interact with Tang.

```
#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.*

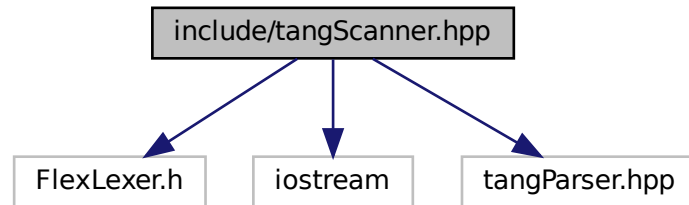
### 6.9.1 Detailed Description

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

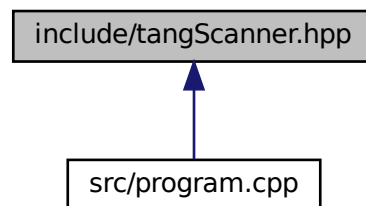
## 6.10 include/tangScanner.hpp File Reference

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\(\)`

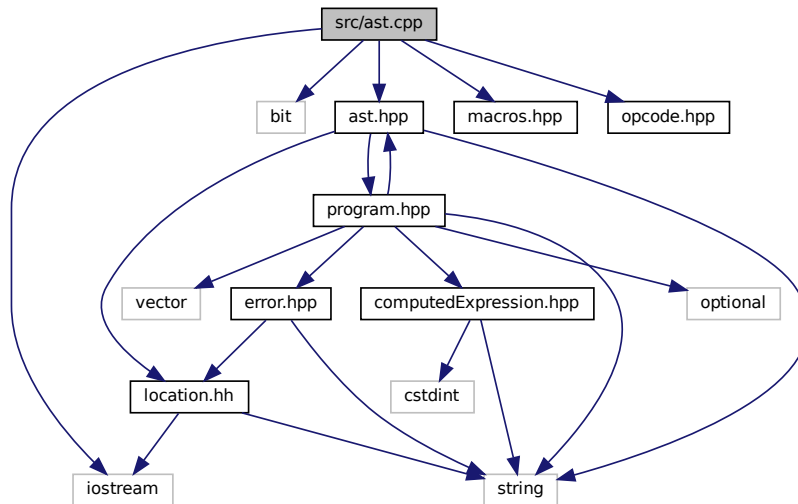
### 6.10.1 Detailed Description

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



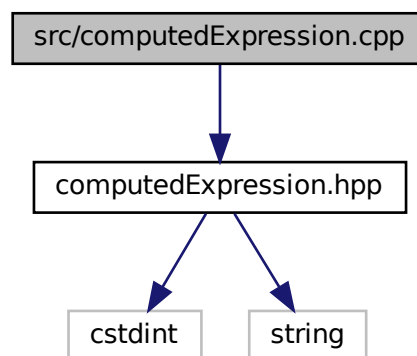
## 6.11 src/ast.cpp File Reference

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



## 6.12 src/computedExpression.cpp File Reference

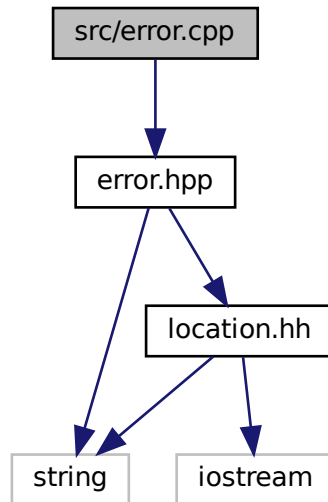
```
#include "computedExpression.hpp"
Include dependency graph for computedExpression.cpp:
```



## 6.13 src/error.cpp File Reference

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

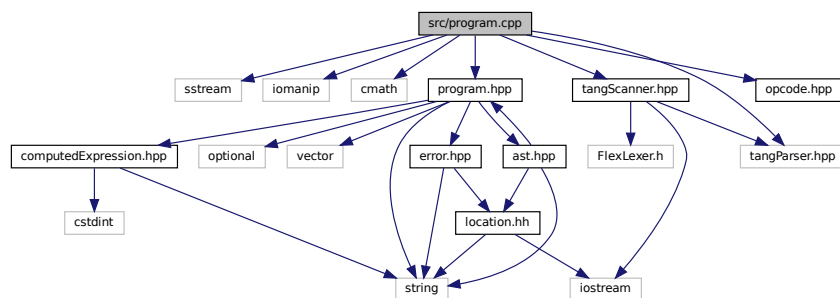
Include dependency graph for error.cpp:



## 6.14 src/program.cpp File Reference

```
#include <sstream>
#include <iomanip>
#include <cmath>
#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.

### 6.14.1 Macro Definition Documentation

#### 6.14.1.1 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

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

#### 6.14.1.2 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.

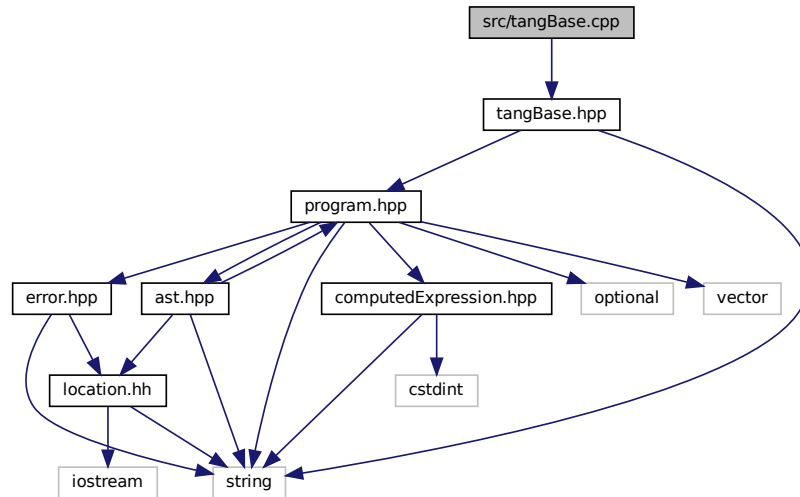
##### Parameters

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

## 6.15 src/tangBase.cpp File Reference

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

Include dependency graph for tangBase.cpp:



# Index

- addBytecode
  - Tang::Program, 25
- AstNode
  - Tang::AstNode, 11
- AstNodeInteger
  - Tang::AstNodeInteger, 14
- build/generated/location.hh, 31
- CodeType
  - Tang::Program, 25
- compileScript
  - Tang::TangBase, 28
- ComputedExpressionInteger
  - Tang::ComputedExpressionInteger, 17
- dump
  - Tang::ComputedExpression, 15
  - Tang::ComputedExpressionInteger, 17
- dumpBytecode
  - Tang::Program, 26
- DUMPPROGRAMCHECK
  - program.cpp, 45
- Error
  - Tang::Error, 20
- execute
  - Tang::Program, 26
- EXECUTEPROGRAMCHECK
  - program.cpp, 45
- get\_next\_token
  - Tang::TangScanner, 30
- getAst
  - Tang::Program, 26
- getCode
  - Tang::Program, 26
- getResult
  - Tang::Program, 27
- include/ast.hpp, 33
- include/computedExpression.hpp, 35
- include/error.hpp, 35
- include/macros.hpp, 37
- include/opcode.hpp, 38
- include/program.hpp, 38
- include/tang.hpp, 40
- include/tangBase.hpp, 40
- include/tangScanner.hpp, 41
- INTEGER
  - opcode.hpp, 38
- location.hh
  - operator<<, 32, 33
- macros.hpp
  - TANG\_UNUSED, 37
- makeCopy
  - Tang::AstNode, 11
  - Tang::AstNodeInteger, 14
  - Tang::ComputedExpression, 15
  - Tang::ComputedExpressionInteger, 17
- Opcode
  - opcode.hpp, 38
- opcode.hpp
  - INTEGER, 38
  - Opcode, 38
- operator<<
  - location.hh, 32, 33
- Program
  - Tang::Program, 25
- program.cpp
  - DUMPPROGRAMCHECK, 45
  - EXECUTEPROGRAMCHECK, 45
- Script
  - Tang::Program, 25
- src/ast.cpp, 43
- src/computedExpression.cpp, 43
- src/error.cpp, 44
- src/program.cpp, 44
- src/tangBase.cpp, 46
- Tang::AstNode, 9
  - AstNode, 11
  - makeCopy, 11
- Tang::AstNodeInteger, 12
  - AstNodeInteger, 14
  - makeCopy, 14
- Tang::ComputedExpression, 15
  - dump, 15
  - makeCopy, 15
- Tang::ComputedExpressionInteger, 16
  - ComputedExpressionInteger, 17
  - dump, 17
  - makeCopy, 17
- Tang::Error, 18
  - Error, 20
- Tang::location, 20
- Tang::position, 22
- Tang::Program, 23

- addBytecode, [25](#)
- CodeType, [25](#)
- dumpBytecode, [26](#)
- execute, [26](#)
- getAst, [26](#)
- getCode, [26](#)
- getResult, [27](#)
- Program, [25](#)
- Script, [25](#)
- Template, [25](#)
- Tang::TangBase, [27](#)
  - compileScript, [28](#)
  - TangBase, [28](#)
- Tang::TangScanner, [28](#)
  - get\_next\_token, [30](#)
  - TangScanner, [29](#)
- TANG\_UNUSED
  - macros.hpp, [37](#)
- TangBase
  - Tang::TangBase, [28](#)
- TangScanner
  - Tang::TangScanner, [29](#)
- Template
  - Tang::Program, [25](#)