

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	10
5.1.2 Constructor & Destructor Documentation	10
5.1.2.1 AstNode()	10
5.2 Tang::AstNodeInteger Class Reference	10
5.2.1 Detailed Description	11
5.2.2 Constructor & Destructor Documentation	11
5.2.2.1 AstNodeInteger()	11
5.3 Tang::Error Class Reference	12
5.3.1 Detailed Description	13
5.3.2 Constructor & Destructor Documentation	13
5.3.2.1 Error()	13
5.4 Tang::location Class Reference	13
5.4.1 Detailed Description	15
5.5 Tang::position Class Reference	15
5.5.1 Detailed Description	16
5.6 Tang::Program Class Reference	16
5.6.1 Detailed Description	17
5.6.2 Member Enumeration Documentation	18
5.6.2.1 CodeType	18
5.6.3 Constructor & Destructor Documentation	18
5.6.3.1 Program()	18
5.6.4 Member Function Documentation	18
5.6.4.1 addBytecode()	18
5.6.4.2 dumpBytecode()	19
5.6.4.3 getAst()	19
5.6.4.4 getCode()	19
5.7 Tang::TangBase Class Reference	20
5.7.1 Detailed Description	20

5.7.2 Constructor & Destructor Documentation	20
5.7.2.1 TangBase()	20
5.7.3 Member Function Documentation	20
5.7.3.1 compileScript()	20
5.8 Tang::TangScanner Class Reference	21
5.8.1 Detailed Description	22
5.8.2 Constructor & Destructor Documentation	22
5.8.2.1 TangScanner()	22
5.8.3 Member Function Documentation	22
5.8.3.1 get_next_token()	22
6 File Documentation	25
6.1 build/generated/location.hh File Reference	25
6.1.1 Detailed Description	26
6.1.2 Function Documentation	26
6.1.2.1 operator<<() [1/2]	27
6.1.2.2 operator<<() [2/2]	27
6.2 include/ast.hpp File Reference	27
6.2.1 Detailed Description	29
6.3 include/error.hpp File Reference	29
6.3.1 Detailed Description	30
6.4 include/macros.hpp File Reference	30
6.4.1 Detailed Description	30
6.4.2 Macro Definition Documentation	30
6.4.2.1 TANG_UNUSED	30
6.5 include/opcode.hpp File Reference	31
6.5.1 Detailed Description	31
6.5.2 Enumeration Type Documentation	31
6.5.2.1 Opcode	31
6.6 include/program.hpp File Reference	32
6.6.1 Detailed Description	33
6.7 include/tang.hpp File Reference	33
6.7.1 Detailed Description	33
6.8 include/tangBase.hpp File Reference	34
6.8.1 Detailed Description	35
6.9 include/tangScanner.hpp File Reference	35
6.9.1 Detailed Description	36
6.10 src/ast.cpp File Reference	36
6.11 src/error.cpp File Reference	37
6.12 src/program.cpp File Reference	37
6.12.1 Macro Definition Documentation	38
6.12.1.1 DUMPPROGRAMCHECK	38

6.13 src/tangBase.cpp File Reference	38
Index	41

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	10
Tang::Error	12
Tang::location	13
Tang::position	15
Tang::Program	16
Tang::TangBase	20
TangTangFlexLexer	
Tang::TangScanner	21

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::AstNodeInteger	An AstNode that represents an integer literal	10
Tang::Error	Used to report any error of the system, whether a syntax (parsing) error or a runtime (execution) error	12
Tang::location	Two points in a source file	13
Tang::position	A point in a source file	15
Tang::Program	Represents a compiled script or template that may be executed	16
Tang::TangBase	The base class for the Tang programming language	20
Tang::TangScanner	The Flex lexer class for the main Tang language	21

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	25
include/ast.hpp	
Define the Tang::AstNode and its associated/derivative classes	27
include/error.hpp	
Define the Tang::Error class used to describe syntax and runtime errors	29
include/macros.hpp	
Contains generic macros	30
include/opcode.hpp	
Declare the Opcodes used in the Bytecode representation of a program	31
include/program.hpp	
Define the Tang::Program class used to compile and execute source code	32
include/tang.hpp	
Header file supplied for use by 3rd party code so that they can easily include all necessary headers	33
include/tangBase.hpp	
Defines the Tang::TangBase class used to interact with Tang	34
include/tangScanner.hpp	
Defines the Tang::TangScanner used to tokenize a Tang script	35
src/ast.cpp	36
src/error.cpp	37
src/program.cpp	37
src/tangBase.cpp	38

Chapter 5

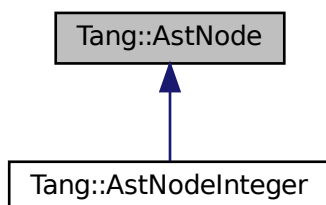
Class Documentation

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:



Public Member Functions

- 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](#).

Protected Member Functions

- [AstNode](#) ([Tang::location](#) loc)
The generic constructor.

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.
------------	---

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

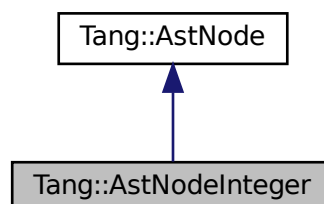
- include/[ast.hpp](#)
- src/[ast.cpp](#)

5.2 Tang::AstNodeInteger Class Reference

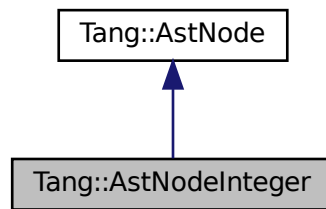
An [AstNode](#) that represents an integer literal.

```
#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](#).

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.

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

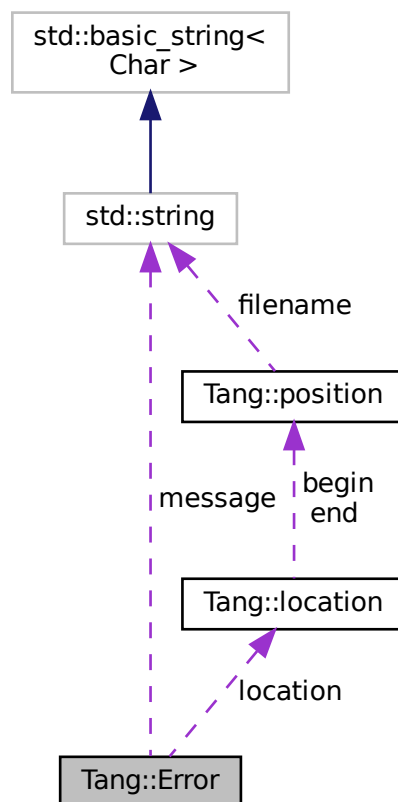
- [include/ast.hpp](#)
- [src/ast.cpp](#)

5.3 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.

5.3.1 Detailed Description

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

5.3.2 Constructor & Destructor Documentation

5.3.2.1 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:

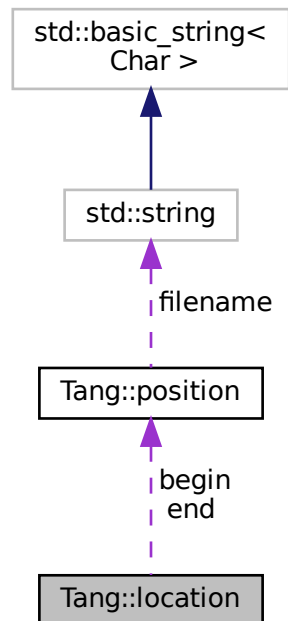
- `include/error.hpp`
- `src/error.cpp`

5.4 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.

5.4.1 Detailed Description

Two points in a source file.

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

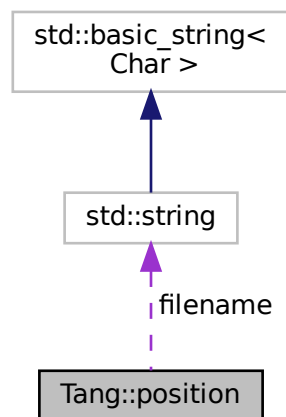
- build/generated/[location.hh](#)

5.5 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.

5.5.1 Detailed Description

A point in a source file.

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

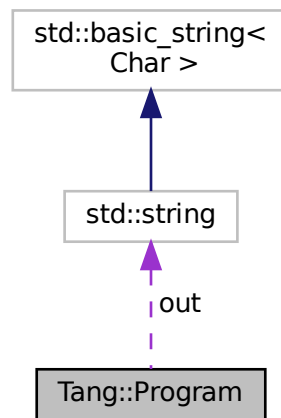
- `build/generated/location.hh`

5.6 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.
- 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.
- void [execute](#) ()
Execute the program.
- void [addBytecode](#) (uint64_t)
Add a uint64_t to the Bytecode.

Public Attributes

- std::string [out](#)
The output of the program, resulting from the program execution.

5.6.1 Detailed Description

Represents a compiled script or template that may be executed.

5.6.2 Member Enumeration Documentation

5.6.2.1 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.

5.6.3 Constructor & Destructor Documentation

5.6.3.1 Program()

```
Tang::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> .

5.6.4 Member Function Documentation

5.6.4.1 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
-----------	----------------------------------

5.6.4.2 dumpBytecode()

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

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

Returns

A string containing the Opcode representation.

5.6.4.3 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.

5.6.4.4 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.

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

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

5.7 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](#).

5.7.1 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).

5.7.2 Constructor & Destructor Documentation

5.7.2.1 TangBase()

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

The constructor.

Isn't it glorious.

5.7.3 Member Function Documentation

5.7.3.1 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:

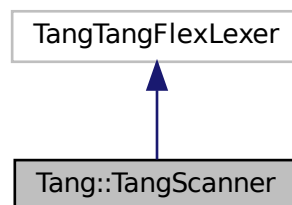
- [include/tangBase.hpp](#)
- [src/tangBase.cpp](#)

5.8 Tang::TangScanner Class Reference

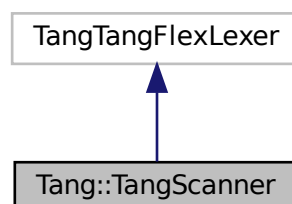
The Flex lexer class for the main Tang language.

```
#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.

5.8.1 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.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 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)

5.8.3 Member Function Documentation

5.8.3.1 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

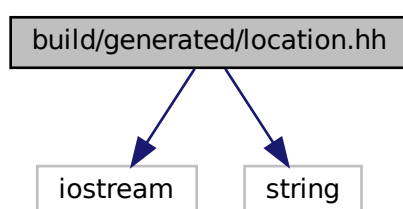
6.1 build/generated/location.hh File Reference

Define the Tang ::location class.

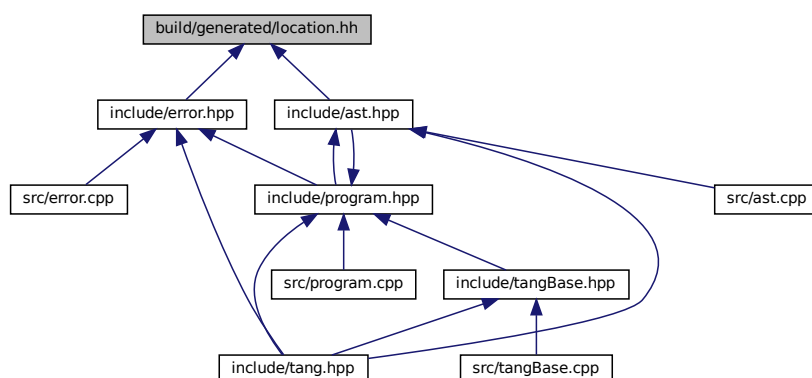
```
#include <iostream>
```

```
#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

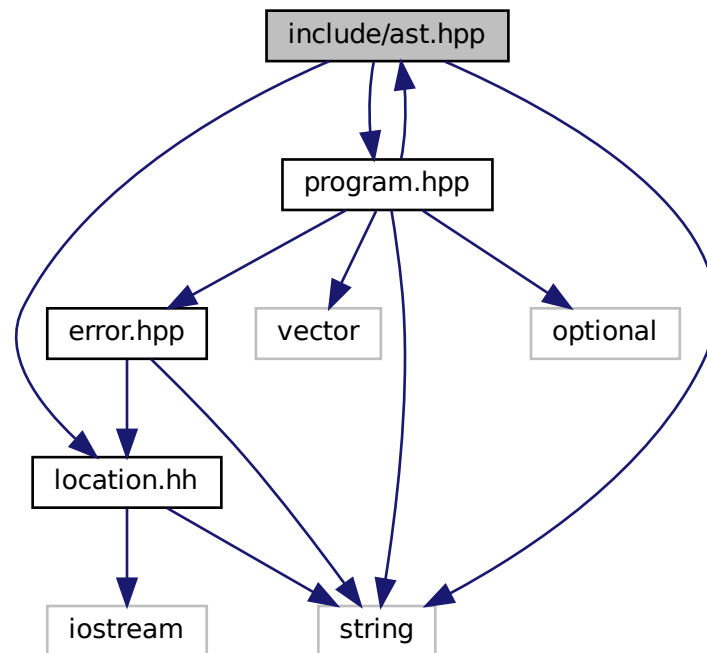
<i>ostr</i>	the destination output stream
<i>pos</i>	a reference to the position to redirect

6.2 include/ast.hpp File Reference

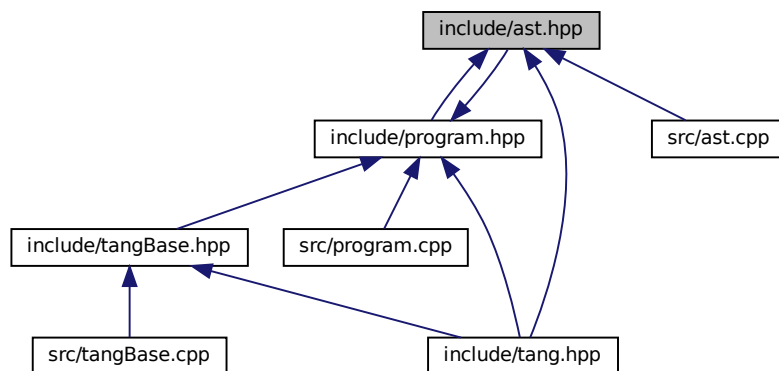
Define the [Tang::AstNode](#) and its associated/derivative classes.

```
#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.

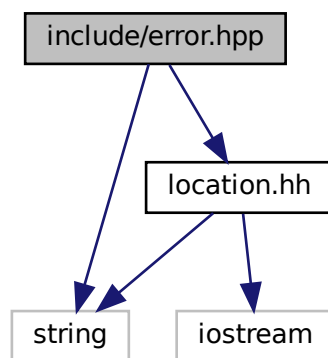
6.2.1 Detailed Description

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

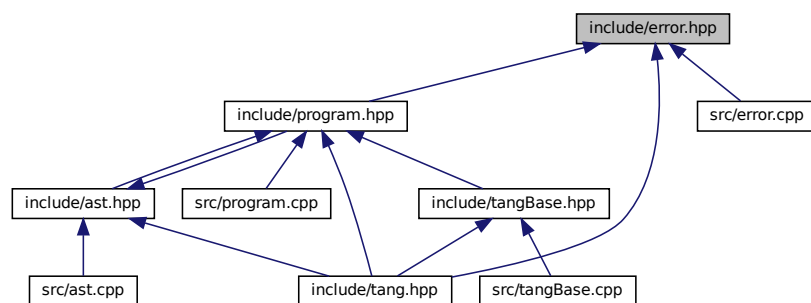
6.3 include/error.hpp File Reference

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.

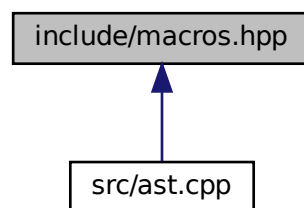
6.3.1 Detailed Description

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

6.4 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.

6.4.1 Detailed Description

Contains generic macros.

6.4.2 Macro Definition Documentation

6.4.2.1 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)) {}`

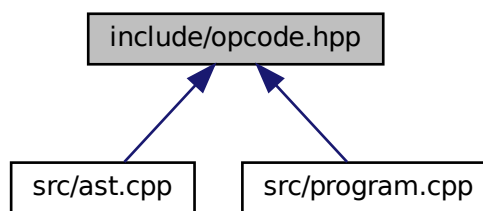
Parameters

<code>x</code>	The argument to be ignored.
----------------	-----------------------------

6.5 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](#) { [OP_INTEGER](#) }

6.5.1 Detailed Description

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

6.5.2 Enumeration Type Documentation

6.5.2.1 Opcode

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

Enumerator

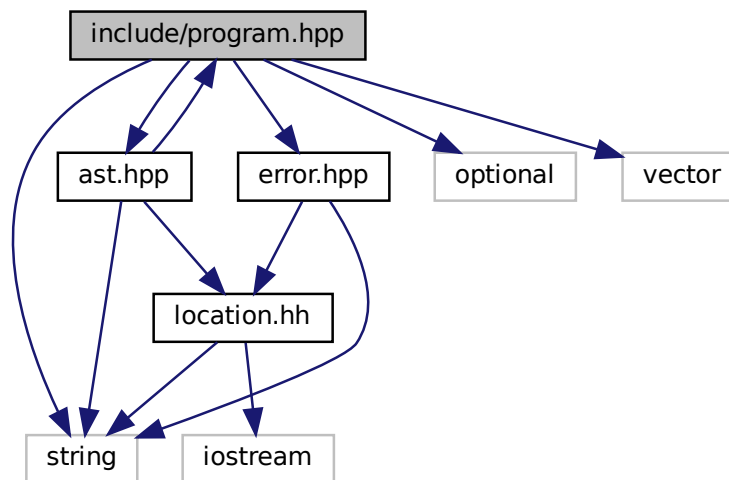
<code>OP_INTEGER</code>	Push an integer onto the stack.
-------------------------	---------------------------------

6.6 include/program.hpp File Reference

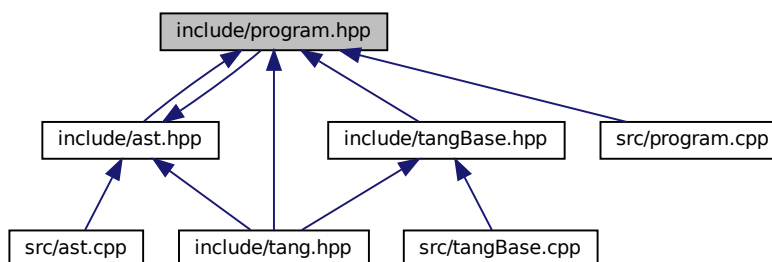
Define the [Tang::Program](#) class used to compile and execute source code.

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

Include 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.

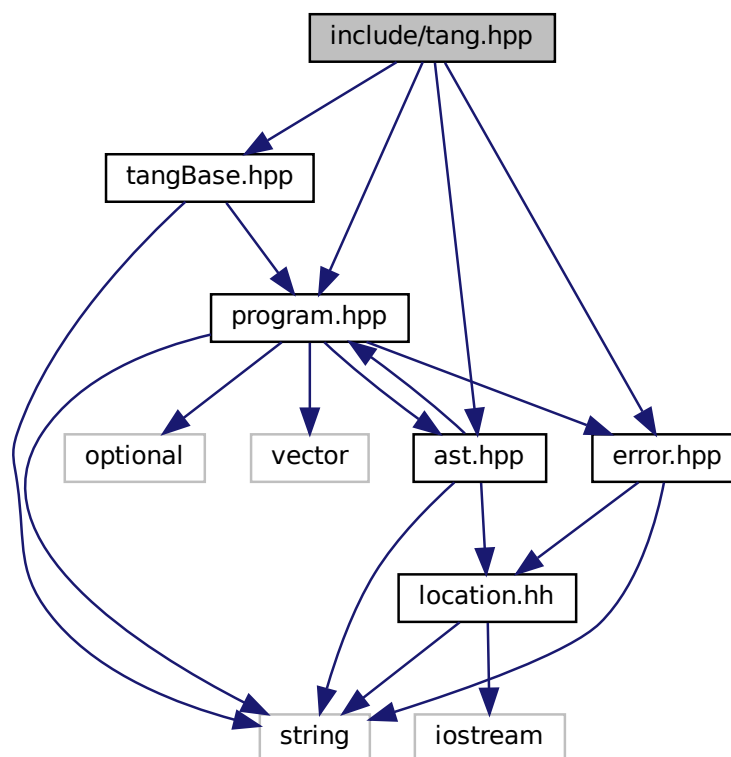
6.6.1 Detailed Description

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

6.7 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:
```



6.7.1 Detailed Description

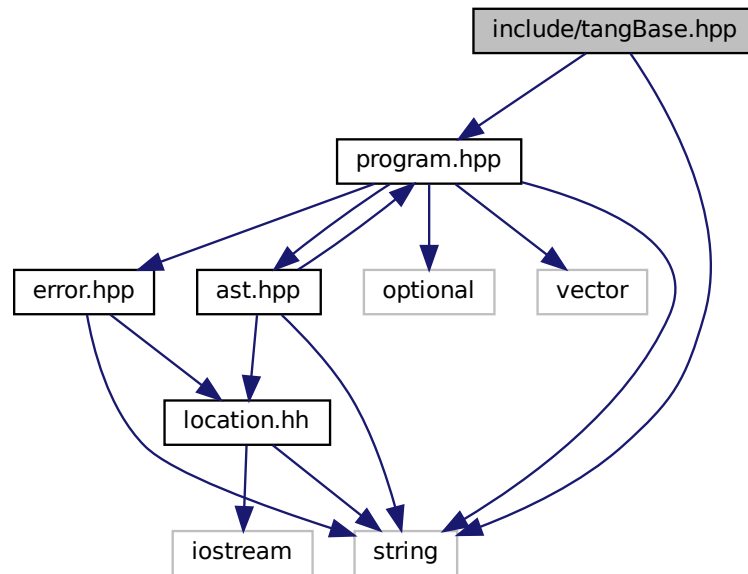
Header file supplied for use by 3rd party code so that they can easily include all necessary headers.

6.8 include/tangBase.hpp File Reference

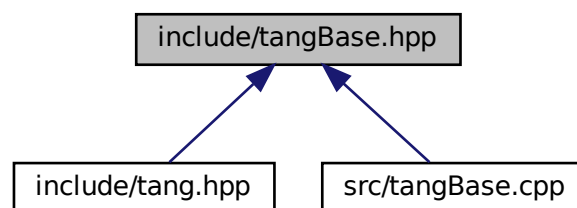
Defines the [Tang::TangBase](#) class used to interact with Tang.

```
#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.

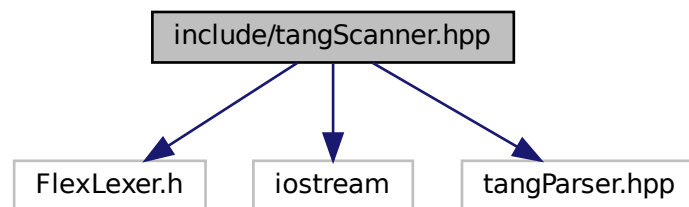
6.8.1 Detailed Description

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

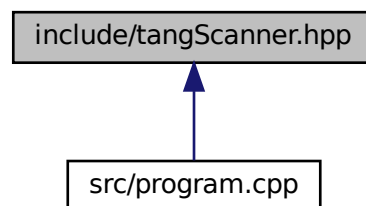
6.9 include/tangScanner.hpp File Reference

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

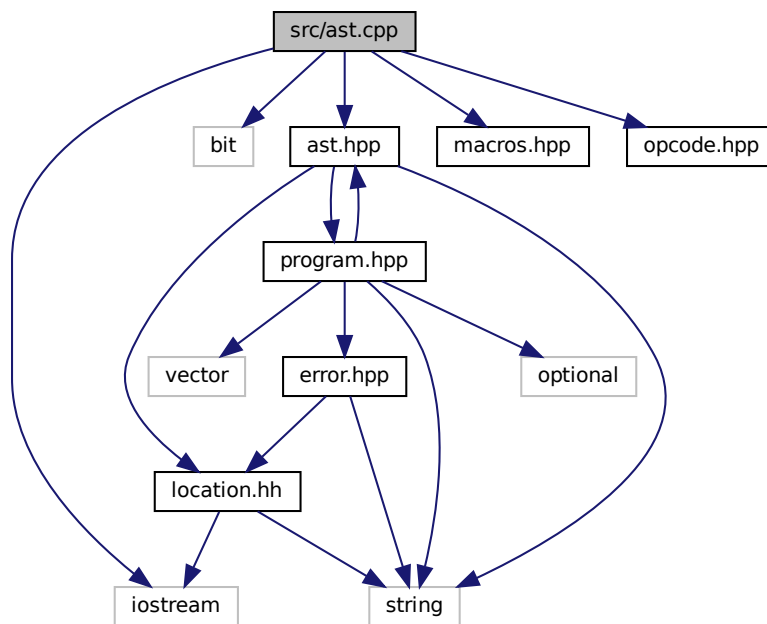
- `#define yyFlexLexer TangTangFlexLexer`
- `#define YY_DECL Tang::TangParser::symbol_type Tang::TangScanner::get_next_token()`

6.9.1 Detailed Description

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

6.10 src/ast.cpp File Reference

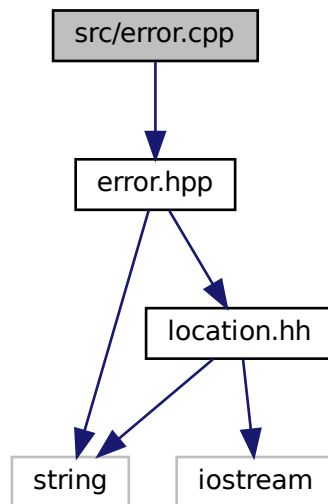
```
#include <iostream>
#include <bit>
#include "ast.hpp"
#include "macros.hpp"
#include "opcode.hpp"
Include dependency graph for ast.cpp:
```



6.11 src/error.cpp File Reference

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

Include dependency graph for error.cpp:



6.12 src/program.cpp File Reference

```
#include <sstream>
```

```
#include <iomanip>
```

```
#include <cmath>
```

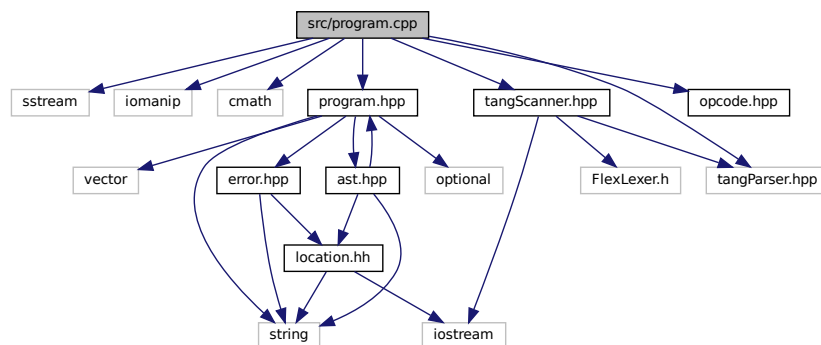
```
#include "program.hpp"
```

```
#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.

6.12.1 Macro Definition Documentation

6.12.1.1 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.

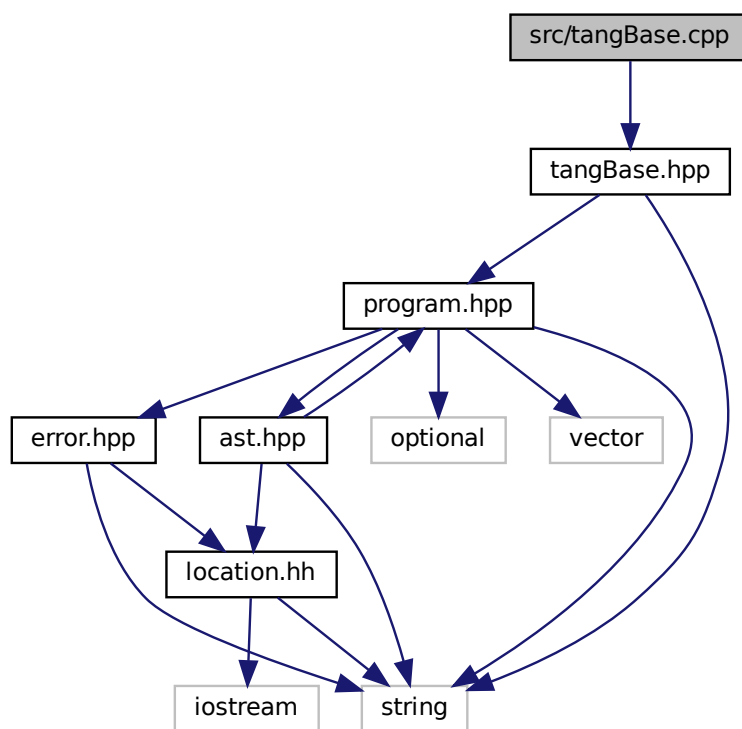
Parameters

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

6.13 src/tangBase.cpp File Reference

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

Include dependency graph for tangBase.cpp:



Index

- addBytecode
 - Tang::Program, [18](#)
- AstNode
 - Tang::AstNode, [10](#)
- AstNodeInteger
 - Tang::AstNodeInteger, [11](#)
- build/generated/location.hh, [25](#)
- CodeType
 - Tang::Program, [18](#)
- compileScript
 - Tang::TangBase, [20](#)
- dumpBytecode
 - Tang::Program, [19](#)
- DUMPPROGRAMCHECK
 - program.cpp, [38](#)
- Error
 - Tang::Error, [13](#)
- get_next_token
 - Tang::TangScanner, [22](#)
- getAst
 - Tang::Program, [19](#)
- getCode
 - Tang::Program, [19](#)
- include/ast.hpp, [27](#)
- include/error.hpp, [29](#)
- include/macros.hpp, [30](#)
- include/opcode.hpp, [31](#)
- include/program.hpp, [32](#)
- include/tang.hpp, [33](#)
- include/tangBase.hpp, [34](#)
- include/tangScanner.hpp, [35](#)
- location.hh
 - operator<<, [26](#), [27](#)
- macros.hpp
 - TANG_UNUSED, [30](#)
- OP_INTEGER
 - opcode.hpp, [31](#)
- Opcode
 - opcode.hpp, [31](#)
- opcode.hpp
 - OP_INTEGER, [31](#)
 - Opcode, [31](#)
- operator<<
 - location.hh, [26](#), [27](#)
- Program
 - Tang::Program, [18](#)
- program.cpp
 - DUMPPROGRAMCHECK, [38](#)
- Script
 - Tang::Program, [18](#)
- src/ast.cpp, [36](#)
- src/error.cpp, [37](#)
- src/program.cpp, [37](#)
- src/tangBase.cpp, [38](#)
- Tang::AstNode, [9](#)
 - AstNode, [10](#)
- Tang::AstNodeInteger, [10](#)
 - AstNodeInteger, [11](#)
- Tang::Error, [12](#)
 - Error, [13](#)
- Tang::location, [13](#)
- Tang::position, [15](#)
- Tang::Program, [16](#)
 - addBytecode, [18](#)
 - CodeType, [18](#)
 - dumpBytecode, [19](#)
 - getAst, [19](#)
 - getCode, [19](#)
 - Program, [18](#)
 - Script, [18](#)
 - Template, [18](#)
- Tang::TangBase, [20](#)
 - compileScript, [20](#)
 - TangBase, [20](#)
- Tang::TangScanner, [21](#)
 - get_next_token, [22](#)
 - TangScanner, [22](#)
- TANG_UNUSED
 - macros.hpp, [30](#)
- TangBase
 - Tang::TangBase, [20](#)
- TangScanner
 - Tang::TangScanner, [22](#)
- Template
 - Tang::Program, [18](#)