

---

# **pygccxml Documentation**

***Release 1.9.1***

**Insight Software Consortium**

**Aug 15, 2017**



---

## Contents

---

|          |                               |            |
|----------|-------------------------------|------------|
| <b>1</b> | <b>Installation</b>           | <b>3</b>   |
| <b>2</b> | <b>Examples</b>               | <b>5</b>   |
| <b>3</b> | <b>Contributing</b>           | <b>7</b>   |
| <b>4</b> | <b>License</b>                | <b>9</b>   |
| <b>5</b> | <b>Test environment</b>       | <b>11</b>  |
| <b>6</b> | <b>Documentation contents</b> | <b>13</b>  |
| <b>7</b> | <b>Indices and tables</b>     | <b>171</b> |
|          | <b>Python Module Index</b>    | <b>173</b> |



The purpose of *pygccxml* is to read a generated xml file and provide a simple framework to navigate C++ declarations, using Python classes.

Using *pygccxml* you can:

- Parse C++ source code
- Create a powerful code generator
- Generate UML diagrams
- Build code analyzers
- ...



# CHAPTER 1

---

## Installation

---

Installation instructions can be found here: [Installation](#)





## CHAPTER 2

---

### Examples

---

The *examples* are a good way to learn how to use *pygccxml*.

*pygccxml* provides a powerful API. If you want to know more about the provided API read the *query interface* document or the *API documentation*.



## CHAPTER 3

---

Contributing

---



## CHAPTER 4

---

### License

---

Boost Software License.



## CHAPTER 5

---

### Test environment

---

*pygccxml* comes with comprehensive unit tests. They are executed on different operating systems, and with different versions of compilers. See the [Travis](#) builds for more details. *pygccxml* is tested under python 2.6, 2.7, 3.2, 3.3, 3.4, 3.5. All in all, *pygccxml* has more than 230 tests.





## Download & Install

### Prerequisite: CastXML

CastXML needs to be installed on your system.

1. If you are on linux or mac, your package manager may already provide a “castxml” package.
2. You can download pre-compiled binaries for [Linux](#), for [OS X](#) and for [Windows](#).
3. You can compile CastXML from source, either with the [SuperBuild](#), or by following the [full install instructions](#).

### Installation of pygccxml

You can use pip to install pygccxml:

```
pip install pygccxml
```

To install from source, you can use the usual procedure:

```
python setup.py install
```

### GCC-XML (Legacy)

These instructions are only here for historical reasons. [GCC-XML](#) was the tool used to generate the xml files before CastXML existed.

**From version v1.8.0 on, pygccxml uses CastXML by default. The support for GCC-XML will finally be dropped in pygccxml v2.0.0.**

There are few different ways to install GCC-XML on your system:

1. Most Linux system provide the “gccxml” package through their package manager.

2. See the [instructions](#) to install GCC-XML from source.

## Examples

### Setting up pygccxml and parsing c/c++ code

#### Parsing a c++ file

This example shows how to setup pygccxml to parse a c++ file, and how to access the declaration tree.

Let's consider the following c++ file (example.hpp):

```
namespace ns{
    int a = 1;
}
```

The following code will show you how to create a configuration for the xml generator (an external tool, either castxml or gccxml), and how to parse the c++ file:

```
from pygccxml import utils
from pygccxml import declarations
from pygccxml import parser

# Find the location of the xml generator (castxml or gccxml)
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
xml_generator_config = parser.xml_generator_configuration_t(
    xml_generator_path=generator_path,
    xml_generator=generator_name)

# The c++ file we want to parse
filename = "example.hpp"

# Parse the c++ file
decls = parser.parse([filename], xml_generator_config)

# Get access to the global namespace
global_namespace = declarations.get_global_namespace(decls)

# Get access to the 'ns' namespace
ns = global_namespace.namespace("ns")
```

#### Parsing a string containing code

This example shows how to setup pygccxml to parse a string containing c++ code, and how to access the declaration tree. Often, pygccxml is used to parse files containing code, but there may be reasons to parse a string (for example for debugging purposes).

The following code will show you how to create a configuration for the xml generator (an external tool, either castxml or gccxml), and how to parse the string containing the c++ code:

```
from pygccxml import utils
from pygccxml import declarations
```

```

from pygccxml import parser

# Find the location of the xml generator (castxml or gccxml)
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
xml_generator_config = parser.xml_generator_configuration_t(
    xml_generator_path=generator_path,
    xml_generator=generator_name)

# Write a string containing some c++ code
code = """
    class MyClass {
        int a;
    };
"""

# Parse the code
decls = parser.parse_string(code, xml_generator_config)

# Get access to the global namespace
global_ns = declarations.get_global_namespace(decls)

```

## First examples

### Variables

This example shows how to find variables and find information about them.

Let's consider the following c++ file:

```

namespace ns{
    int a = 1;
    int b = 2;
    double c = 3.0;
}

```

The following code can be use to find the variable named “c” using different strategies, and to print information about it:

```

from pygccxml import utils
from pygccxml import declarations
from pygccxml import parser

# Find out the c++ parser
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
xml_generator_config = parser.xml_generator_configuration_t(
    xml_generator_path=generator_path,
    xml_generator=generator_name)

# The c++ file we want to parse
filename = "example.hpp"

decls = parser.parse([filename], xml_generator_config)

```

```
global_namespace = declarations.get_global_namespace(decls)
ns = global_namespace.namespace("ns")

# The variables() method will return a list of variables.
# We know that the c variable is the third one in the list:
c = ns.variables()[2]
print("My name is: " + c.name)
print("My type is: " + str(c.decl_type))
print("My value is: " + c.value)

# Of course you can also loop over the list and look for the right name
for var in ns.variables():
    if var.name == "c":
        print("My name is: " + var.name)
        print("My type is: " + str(var.decl_type))
        print("My value is: " + var.value)

# One way to get a variable is to use the variable() method and
# a lambda function. This is the most flexible way as you can implement
# your own lambda function to filter out variables following your
# specific criteria.
c = ns.variable(lambda v: v.name == "c")
print("My name is: " + c.name)
print("My type is: " + str(c.decl_type))
print("My value is: " + c.value)
```

## Searching for a declaration (using a loop)

This example shows how to search for a specific declaration using a loop on the declarations tree.

Let's consider the following c++ file (example.hpp):

```
namespace ns{
    int a = 1;
    int b = 2;
    double c = 3.0;

    double func2(double a) {
        double b = a + 2.0;
        return b;
    }
}
```

The following code will show you how to loop on the tree and find a declaration

```
from pygccxml import utils
from pygccxml import declarations
from pygccxml import parser

# Find the location of the xml generator (castxml or gccxml)
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
xml_generator_config = parser.xml_generator_configuration_t(
    xml_generator_path=generator_path,
    xml_generator=generator_name)
```

```

# The c++ file we want to parse
filename = "example.hpp"

# Parse the c++ file
decls = parser.parse([filename], xml_generator_config)

global_namespace = declarations.get_global_namespace(decls)

ns_namespace = global_namespace.namespace("ns")

int_type = declarations.cpptypes.int_t()
double_type = declarations.cpptypes.double_t()

for decl in ns_namespace.declarations:
    print(decl)

# This prints all the declarations in the namespace declaration tree:
# ns::a [variable]
# ns::b [variable]
# ns::c [variable]
# double ns::func2(double a) [free function]

# Let's search for specific declarations
for decl in ns_namespace.declarations:
    if decl.name == "b":
        print(decl)
    if isinstance(decl, declarations.free_function_t):
        print(decl)

# This prints:
# ns::b [variable]
# double ns::func2(double a) [free function]

```

## Searching for a declaration (using matchers)

This example shows how to search for a specific declaration using different criteria.

Let's consider the following c++ file (example.hpp):

```

namespace ns{
    int a = 1;
    int b = 2;
    double c = 3.0;

    int func1(int a) {
        int b = a + 2;
        return b;
    }

    double func2(double a) {
        double b = a + 2.0;
        return b;
    }

    double func3(double a) {
        double b = a + 3.0;
        return b;
    }
}

```

```
}  
}
```

The following code will show you how to search for functions and variables

```
from pygccxml import utils  
from pygccxml import declarations  
from pygccxml import parser  
  
# Find the location of the xml generator (castxml or gccxml)  
generator_path, generator_name = utils.find_xml_generator()  
  
# Configure the xml generator  
xml_generator_config = parser.xml_generator_configuration_t(  
    xml_generator_path=generator_path,  
    xml_generator=generator_name)  
  
# The c++ file we want to parse  
filename = "example.hpp"  
  
# Parse the c++ file  
decls = parser.parse([filename], xml_generator_config)  
  
global_namespace = declarations.get_global_namespace(decls)  
  
ns_namespace = global_namespace.namespace("ns")  
  
int_type = declarations.cpptypes.int_t()  
double_type = declarations.cpptypes.double_t()  
  
# Search for the function by name  
criteria = declarations.calldef_matcher(name="func1")  
func1 = declarations.matcher.get_single(criteria, ns_namespace)  
  
# Search for functions which return an int  
criteria = declarations.calldef_matcher(return_type="int")  
func2 = declarations.matcher.get_single(criteria, ns_namespace)  
  
# Search for functions which return an int, using the cpptypes class  
criteria = declarations.calldef_matcher(return_type=int_type)  
func3 = declarations.matcher.get_single(criteria, ns_namespace)  
  
print(func1)  
print(func2)  
print(func3)  
  
# This prints:  
# int ns::func1(int a) [free function]  
# int ns::func1(int a) [free function]  
# int ns::func1(int a) [free function]  
  
# Search for functions which return a double. Two functions will be found  
criteria = declarations.calldef_matcher(return_type=double_type)  
func4 = declarations.matcher.find(criteria, ns_namespace)  
  
print(len(func4))  
print(func4[0])  
print(func4[1])
```

```

# This prints:
# 2
# double ns::func2(double a) [free function]
# double ns::func3(double a) [free function]

# Finally, look for variables by name and by type
criteria = declarations.variable_matcher(name="a")
var_a1 = declarations.matcher.find(criteria, ns_namespace)

criteria = declarations.variable_matcher(decl_type=int_type)
var_a2 = declarations.matcher.find(criteria, ns_namespace)

print(var_a1[0])
print(var_a2[0])
print(var_a2[1])

# This prints:
# ns::a [variable]
# ns::a [variable]
# ns::b [variable]

```

## Comparing two declarations

This example shows how two declarations can be compared.

Let's consider the following c++ file (example.hpp):

```

namespace ns{
    void func1(int a) {
        int b = a;
    }

    void func2(int a) {
        int b = a;
    }
}

```

The following code will show you how to search for two functions, using different ways. Both declarations are then compared.

```

from pygccxml import utils
from pygccxml import declarations
from pygccxml import parser

# Find the location of the xml generator (castxml or gccxml)
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
xml_generator_config = parser.xml_generator_configuration_t(
    xml_generator_path=generator_path,
    xml_generator=generator_name)

# The c++ file we want to parse
filename = "example.hpp"

# Parse the c++ file

```

```
decls = parser.parse([filename], xml_generator_config)

global_namespace = declarations.get_global_namespace(decls)

ns_namespace = global_namespace.namespace("ns")

# Search for the function called func1
criteria = declarations.calldef_matcher(name="func1")
func1a = declarations.matcher.get_single(criteria, ns_namespace)

# Search for the function called func2
criteria = declarations.calldef_matcher(name="func2")
func2a = declarations.matcher.get_single(criteria, ns_namespace)

# You can also write a loop on the declaration tree
func1b = None
for decl in ns_namespace.declarations:
    if decl.name == "func1":
        func1b = decl

# The declarations can be compared (prints (True, False))
print(func1a == func1b, func1a == func2a)
```

## Functions and arguments

This example shows how to work with function arguments

Let's consider the following c++ file:

```
#include <iostream>
using namespace std;

namespace ns{
    int myFunction(int a, const std::string& x1) {
        return a + 1;
    }
}
```

The following code can be used to find the different arguments of a function and do some basic operations on them:

```
from pygccxml import utils
from pygccxml import declarations
from pygccxml import parser

# Find out the c++ parser
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
xml_generator_config = parser.xml_generator_configuration_t(
    xml_generator_path=generator_path,
    xml_generator=generator_name)

# The c++ file we want to parse
filename = "example.hpp"

decls = parser.parse([filename], xml_generator_config)
global_namespace = declarations.get_global_namespace(decls)
```



```

ns = global_namespace.namespace("ns")

# Use the free_functions method to find our function
func = ns.free_function(name="myFunction")

# There are two arguments:
print(len(func.arguments))

# We can loop over them and print some information:
for arg in func.arguments:
    print(
        arg.name,
        str(arg.decl_type),
        declarations.is_std_string(arg.decl_type),
        declarations.is_reference(arg.decl_type))

```

## Nested types

This example shows how to work with types.

Let's consider the following c++ file:

```

namespace ns{
    const int a = 0;
    const volatile int *b = 0;
}

```

The following code allows you to extract information about the types of variables:

```

from pygccxml import utils
from pygccxml import declarations
from pygccxml import parser

# Find out the c++ parser
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
xml_generator_config = parser.xml_generator_configuration_t(
    xml_generator_path=generator_path,
    xml_generator=generator_name)

# The c++ file we want to parse
filename = "example.hpp"

decls = parser.parse([filename], xml_generator_config)
global_namespace = declarations.get_global_namespace(decls)
ns = global_namespace.namespace("ns")

a = ns.variables()[0]

print("My name is: " + a.name)
# > My name is: a

print("My type is: " + str(a.decl_type))
# > My type is: int const

# If we print real python type:

```

```
print("My type is : " + str(type(a.decl_type)))
# > My type is: <class 'pygccxml.declarations.cpptypes.const_t'>

# Types are nested in pygccxml. This means that you will get information
# about the first type only. You can access the "base" type by removing
# the const part:
print("My base type is: " + str(type(declarations.remove_const(a.decl_type))))
# > My base type is: <class 'pygccxml.declarations.cpptypes.int_t'>

# You use the is_const function to check for a type:
print("Is 'a' a const ?: " + str(declarations.is_const(a.decl_type)))
# > Is 'a' a const ?: True

# A more complex example with variable b:
b = ns.variables()[1]
print("My type is: " + str(type(b.decl_type)))
# > My type is: <class 'pygccxml.declarations.cpptypes.pointer_t'>
print("My type is: " + str(type(
    declarations.remove_const(
        declarations.remove_volatile(
            declarations.remove_pointer(b.decl_type)))))
# > My type is: <class 'pygccxml.declarations.cpptypes.int_t'>

# The declarations module contains much more methods allowing you to
# navigate the nested types list.
```

## Explicit and implicit class declarations

Even if a class has no explicit constructor, pygccxml will provide a constructor declaration. This is due to CastXML and GCC-XML generating implicit constructors (for example copy constructors) in their XML output. The same thing holds for assignment operators and destructors.

To be able to discriminate between the different types of declarations, the `decl.is_artificial` attribute can be used.

Let's consider the following c++ file (example.hpp):

```
namespace ns{
    class Test {
    public:
        Test(); // This is the constructor
    };
}
```

In this example, the constructor is explicitly defined. The declaration tree will contain two constructors. The first one is the one we defined explicitly, and is not marked as artificial. The second one is the copy constructor, which was implicitly added, and is marked as artificial.

```
from pygccxml import utils
from pygccxml import declarations
from pygccxml import parser

# Find out the c++ parser
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
xml_generator_config = parser.xml_generator_configuration_t(
```

```

    xml_generator_path=generator_path,
    xml_generator=generator_name)

# The c++ file we want to parse
filename = "example.hpp"

decls = parser.parse([filename], xml_generator_config)
global_namespace = declarations.get_global_namespace(decls)
ns = global_namespace.namespace("ns")

# We have just one declaration in ns, which is our Test class:
classTest = ns.declarations[0]
print(classTest.name, type(classTest))
# > 'Test', <class 'pygccxml.declarations.class_declaration.class_t'

# Loop over the two constructors:
for constructor in classTest.constructors():
    print(str(constructor), constructor.is_artificial)
# > ns::Test::Test() [constructor], False
# > ns::Test::Test(ns::Test const & arg0) [constructor], True

```

## Compound types

A type is a `compound_t` type (in pygccxml) if it is one of the following: *volatile\_t*, *restrict\_t*, *const\_t*, *pointer\_t*, *reference\_t*, *elaborated\_t*, *array\_t* or *member\_variable\_type\_t*.

The exact c++ definition of compound types embraces more types, but for different reasons (mostly legacy), the definition in pygccxml is slightly different.

Let's consider the following c++ file:

```

int const c1 = 0;
const int c2 = 0;

volatile const int cv1 = 0;
const volatile int cv2 = 0;

const int * const cptr1 = 0;

```

The following code will show what to expect from compound types, how they are chained, and how their order is defined in pygccxml.

```

from pygccxml import utils
from pygccxml import declarations
from pygccxml import parser

# Find out the c++ parser
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
xml_generator_config = parser.xml_generator_configuration_t(
    xml_generator_path=generator_path,
    xml_generator=generator_name)

# The c++ file we want to parse
filename = "example.hpp"

```

```
decls = parser.parse([filename], xml_generator_config)
global_namespace = declarations.get_global_namespace(decls)

c1 = global_namespace.variable("c1")
print(str(c1), type(c1))
# > 'c1 [variable]', <class 'pygccxml.declarations.variable.variable_t'>

print(str(c1.decl_type), type(c1.decl_type))
# > 'int const', <class 'pygccxml.declarations.cpptypes.const_t'>

base = declarations.remove_const(c1.decl_type)
print(str(base), type(base))
# > 'int', <class 'pygccxml.declarations.cpptypes.int_t'>

c2 = global_namespace.variable("c2")
print(str(c2.decl_type), type(c2.decl_type))
# > 'int const', <class 'pygccxml.declarations.cpptypes.const_t'>
# Even if the declaration was defined as 'const int', pygccxml will always
# output the const qualifier (and some other qualifiers) on the right hand
# side (by convention).

cv1 = global_namespace.variable("cv1")
print(str(cv1.decl_type), type(cv1.decl_type))
# > 'int const volatile', <class 'pygccxml.declarations.cpptypes.volatile_t'>

# Remove one level:
base = declarations.remove_volatile(cv1.decl_type)
print(str(base), type(base))
# > 'int const', <class 'pygccxml.declarations.cpptypes.const_t'>

# Remove the second level:
base = declarations.remove_const(base)
print(str(base), type(base))
# > 'int', <class 'pygccxml.declarations.cpptypes.int_t'>

# We can also directly do this in one step:
base = declarations.remove_cv(cv1.decl_type)
print(str(base), type(base))
# > 'int', <class 'pygccxml.declarations.cpptypes.int_t'>

# As for consts, the const and volatile are on the right hand side
# (by convention), and always in the same order
cv2 = global_namespace.variable("cv2")
print(str(cv2.decl_type), type(cv2.decl_type))
# > 'int const volatile', <class 'pygccxml.declarations.cpptypes.volatile_t'>

# And a last example with a pointer_t:
cptr1 = global_namespace.variable("cptr1")
print(str(cptr1.decl_type), type(cptr1.decl_type))
# > 'int const * const', <class 'pygccxml.declarations.cpptypes.const_t'>)

base = declarations.remove_const(cptr1.decl_type)
print(str(base), type(base))
# > 'int const *', <class 'pygccxml.declarations.cpptypes.pointer_t'>

base = declarations.remove_pointer(base)
print(str(base), type(base))
# > 'int const', <class 'pygccxml.declarations.cpptypes.const_t'>)
```

```
base = declarations.remove_const(base)
print(str(base), type(base))
# > 'int', <class 'pygccxml.declarations.cpptypes.int_t'>
```

## C++ Templates

pygccxml has minimal support for c++ templates, but there is some information that can be extracted from templated declarations.

Let's consider the following c++ file (example.hpp):

```
namespace ns {

struct B {
    struct D { bool d; };
};
struct D {};

template <typename T1, typename T2>
struct T {};

T<B::D, bool> function();

}
```

This example show how to extract template parameters from the template declaration.

```
from pygccxml import utils
from pygccxml import declarations
from pygccxml import parser

# Find out the c++ parser
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
xml_generator_config = parser.xml_generator_configuration_t(
    xml_generator_path=generator_path,
    xml_generator=generator_name)

# The c++ file we want to parse
filename = "example.hpp"

decls = parser.parse([filename], xml_generator_config)
global_namespace = declarations.get_global_namespace(decls)
ns = global_namespace.namespace("ns")

class_t_decl = []
for d in ns.declarations:
    if isinstance(d, declarations.class_declaration_t):
        class_declaration_t = d
    if isinstance(d, declarations.class_t):
        class_t_decl.append(d)
    if isinstance(d, declarations.free_function_t):
        free_function_t_decl = d

print(class_t_decl[0])
```

```
# > ns::B [struct]

print(class_t_decl[1])
# > ns::D [struct]

print(class_declaration_t)
# > ns::T<ns::B::D, bool> [class declaration]

print(free_function_t_decl)
# > ns::T<ns::B::D, bool> ns::function() [free function]

print(declarations.templates.is_instantiation(class_declaration_t.name))
# > True

name, parameter_list = declarations.templates.split(class_declaration_t.name)
print(name, parameter_list)
# > 'T', ['ns::B::D', 'bool']
```

## Advanced examples

### Elaborated type specifiers

Elaborated type specifiers are one of these four possibilities: *class*, *struct*, *union* or *enum*.

In C++ they can often be skipped (but may be useful; see [this interesting topic](#) for example). In C code they are mandatory.

Let's consider the following c++ file:

```
class A {};
```

A a1;

```
class A a2;
```

void function(A arg1, class A arg2);

The following code will show how the elaborated type specifiers are treated in pygccxml. Please note that this feature is only available since recent versions of *CastXML* (Mar 1, 2017), and a special flag needs to be passed to pygccxml to make this work (castxml\_epic\_version=1).

```
from pygccxml import utils
from pygccxml import declarations
from pygccxml import parser

# Find out the c++ parser
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
xml_generator_config = parser.xml_generator_configuration_t(
    xml_generator_path=generator_path,
    xml_generator=generator_name,
    castxml_epic_version=1)

# The c++ file we want to parse
filename = this_module_dir_path + "/" + filename
```

```

decls = parser.parse([filename], xml_generator_config)
global_namespace = declarations.get_global_namespace(decls)

a1 = global_namespace.variable("a1")
print(str(a1.decl_type), type(a1.decl_type))
# > 'A', <class 'pygccxml.declarations.cpptypes.declared_t'>

print(declarations.is_elaborated(a1.decl_type))
# > False

a2 = global_namespace.variable("a2")
print(str(a2.decl_type), type(a2.decl_type))
# > 'class ::A', <class 'pygccxml.declarations.cpptypes.elaborated_t'>

print(declarations.is_elaborated(a2.decl_type))
# > True

base = declarations.remove_elaborated(a2.decl_type)
print(str(base), type(base))
# > 'A', <class 'pygccxml.declarations.cpptypes.declared_t'>

# The same can be done with function arguments:
fun = global_namespace.free_function("function")
print(type(fun.arguments[0].decl_type), type(fun.arguments[1].decl_type))
# > <class 'pygccxml.declarations.cpptypes.declared_t'>,
# <class 'pygccxml.declarations.cpptypes.elaborated_t'>

```

## Function pointers

This example shows how to work with function pointers.

Let's consider the following c++ file:

```

// A function pointer
void (*myFuncPointer)(int, double);

```

The following code allows you to extract information about the function pointer:

```

from pygccxml import utils
from pygccxml import declarations
from pygccxml import parser

# Find out the c++ parser
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
xml_generator_config = parser.xml_generator_configuration_t(
    xml_generator_path=generator_path,
    xml_generator=generator_name)

# The c++ file we want to parse
filename = "example.hpp"

decls = parser.parse([filename], xml_generator_config)
global_namespace = declarations.get_global_namespace(decls)

function_ptr = global_namespace.variables()[0]

```

```
# Print the name of the function pointer
print(function_ptr.name)
# > myFuncPointer

# Print the type of the declaration
print(function_ptr.decl_type)
# > void (*)( int,double )

# Print the real type of the declaration (it's just a pointer)
print(type(function_ptr.decl_type))
# > <class 'pygccxml.declarations.cpptypes.pointer_t'>

# Check if this is a function pointer
print(declarations.is_calldef_pointer(function_ptr.decl_type))
# > True

# Remove the pointer part, to access the function's type
f_type = declarations.remove_pointer(function_ptr.decl_type)

# Print the type
print(type(f_type))
# > <class 'pygccxml.declarations.cpptypes.free_function_type_t'>

# Print the return type and the arguments of the function
print(f_type.return_type)
# > void

# Print the return type and the arguments
print(str(f_type.arguments_types[0]), str(f_type.arguments_types[1]))
# > int, double
```

## Caching

This example shows how to use caching. This can be useful for big projects where you don't want the c++ to be parsed again and again.

Let's consider the following c++ file:

```
namespace ns{
    int a = 1;
}
```

To enable caching, you can use the following code:

```
from pygccxml import utils
from pygccxml import declarations
from pygccxml import parser

# Find out the c++ parser
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
xml_generator_config = parser.xml_generator_configuration_t(
    xml_generator_path=generator_path,
    xml_generator=generator_name)
```



```
# The c++ file we want to parse
filename = "example.hpp"

file_config = parser.file_configuration_t(
    data=filename,
    content_type=parser.CONTENT_TYPE.CACHED_SOURCE_FILE)

project_reader = parser.project_reader_t(xml_generator_config)
decls = project_reader.read_files(
    [file_config],
    compilation_mode=parser.COMPILATION_MODE.FILE_BY_FILE)

global_namespace = declarations.get_global_namespace(decls)

value = global_namespace.namespace("ns")
print("My name is: " + value.name)
```

The first time you run this example, the c++ file will be read and a xml file will be generated:

INFO Creating xml file “example.hpp.xml” from source file “example.hpp” ... INFO Parsing xml file “example.hpp.xml” ... My name is: ns

The second time you run the example the xml file will not be regenerated:

INFO Parsing xml file “example.hpp.xml” ... My name is: ns

Of course the performance gain will be small for this example, but can be interesting for bigger projects.

## Print all declarations

This example prints all declarations found in *example.hpp* file.

For every class, it prints it’s base and derived classes.

The example consists from few files:

### C++ header file

```
#ifndef example_hpp_12_10_2006
#define example_hpp_12_10_2006

namespace unittests{

struct test_results{

    enum status{ ok, fail, error };

    void update( const char* test_name, status result );
};

struct test_case{

    test_case( const char* test_case_name );

    virtual void set_up() {}
}
```

```
virtual void tear_down() {}

virtual void run() = 0;

private:
    const char* m_name;
};

class test_container;

struct test_suite : public test_case{

    test_suite( const char* name, const test_container& tests );

    void run();

    const test_results& get_results() const
    { return m_results; }

private:
    test_container* m_tests;
    test_results m_results;
};

}

#endif//example_hpp_12_10_2006
```

## GCC-XML generated file

```
<?xml version="1.0"?>
<GCC_XML cvs_revision="1.127">
  <Namespace id="_1" name="::" members="_3 _4 _5 _6 _2 " mangled="_Z2::" demangled="::"
  ↪"/>
  <Namespace id="_2" name="std" context="_1" members="" mangled="_Z3std" demangled=
  ↪"std"/>
  <Variable id="_3" name="_ZTIN9unittests10test_suiteE" type="_7c" context="_1"
  ↪location="f0:35" file="f0" line="35" extern="1" artificial="1"/>
  <Namespace id="_4" name="unittests" context="_1" members="_9 _10 _11 _12 " mangled=
  ↪"_Z9unittests" demangled="unittests"/>
  <Variable id="_5" name="_ZTIN9unittests9test_caseE" type="_13c" context="_1"
  ↪location="f0:19" file="f0" line="19" extern="1" artificial="1"/>
  <Namespace id="_6" name="__cxxabiv1" context="_1" members="" mangled="_Z10__cxxabiv1
  ↪" demangled="__cxxabiv1"/>
  <Struct id="_7" name="__si_class_type_info_pseudo" context="_1" mangled="27__si_
  ↪class_type_info_pseudo" demangled="__si_class_type_info_pseudo" location="f1:0"
  ↪file="f1" line="0" size="96" align="32" members=""/>
  <CvQualifiedType id="_7c" type="_7" const="1"/>
  <Struct id="_9" name="test_suite" context="_4" mangled="N9unittests10test_suiteE"
  ↪demangled="unittests::test_suite" location="f0:35" file="f0" line="35" artificial="1"
  ↪size="128" align="32" members="_15 _16 _17 _18 _19 _20 _21 _22 " bases="_12 ">
    <Base type="_12" access="public" virtual="0" offset="0"/>
  </Struct>
  <Struct id="_10" name="test_container" context="_4" incomplete="1" mangled=
  ↪"N9unittests14test_containerE" demangled="unittests::test_container" location="f0:33"
  ↪file="f0" line="33" artificial="1" align="8"/>
```

```

<Struct id="_11" name="test_results" context="_4" mangled="N9unittests12test_
→resultsE" demangled="unittests::test_results" location="f0:12" file="f0" line="12"
→artificial="1" size="8" align="8" members="_23 _24 _25 _26 _27 _28 " bases=""/>
  <Struct id="_12" name="test_case" context="_4" abstract="1" mangled=
→"N9unittests9test_caseE" demangled="unittests::test_case" location="f0:19" file="f0
→" line="19" artificial="1" size="64" align="32" members="_29 _30 _31 _32 _33 _34 _
→35 _36 " bases=""/>
  <Struct id="_13" name="__class_type_info_pseudo" context="_1" mangled="24__class_
→type_info_pseudo" demangled="__class_type_info_pseudo" location="f1:0" file="f1"
→line="0" size="64" align="32" members=""/>
  <CvQualifiedType id="_13c" type="_13" const="1"/>
  <Field id="_15" name="m_tests" type="_37" offset="64" context="_9" access="private"
→location="f0:45" file="f0" line="45"/>
  <Field id="_16" name="m_results" type="_11" offset="96" context="_9" access="private
→" location="f0:46" file="f0" line="46"/>
  <Destructor id="_17" name="test_suite" artificial="1" throw="" context="_9" access=
→"public" mangled="_ZN9unittests10test_suiteD1Ev *INTERNAL* " demangled=
→"unittests::test_suite::~~test_suite()" location="f0:35" file="f0" line="35" endl=
→"35" inline="1">
  </Destructor>
  <OperatorMethod id="_18" name="" returns="_38" artificial="1" throw="" context="_9
→" access="public" mangled="_ZN9unittests10test_suiteaSERKS0_" demangled=
→"unittests::test_suite::operator=(unittests::test_suite const&)" location="f0:35
→" file="f0" line="35" endl="35" inline="1">
    <Argument type="_39" location="f0:35" file="f0" line="35"/>
  </OperatorMethod>
  <Constructor id="_19" name="test_suite" artificial="1" throw="" context="_9" access=
→"public" mangled="_ZN9unittests10test_suiteC1ERKS0_ *INTERNAL* " demangled=
→"unittests::test_suite::test_suite(unittests::test_suite const&)" location=
→"f0:35" file="f0" line="35" endl="35" inline="1">
    <Argument type="_39" location="f0:35" file="f0" line="35"/>
  </Constructor>
  <Constructor id="_20" name="test_suite" explicit="1" context="_9" access="public"
→mangled="_ZN9unittests10test_suiteC1EPKcRKNS_14test_containerE *INTERNAL* "
→demangled="unittests::test_suite::test_suite(char const*, unittests::test_container
→const&)" location="f0:37" file="f0" line="37" extern="1">
    <Argument name="name" type="_40" location="f0:37" file="f0" line="37"/>
    <Argument name="tests" type="_41" location="f0:37" file="f0" line="37"/>
  </Constructor>
  <Method id="_21" name="run" returns="_42" virtual="1" overrides="_36 " context="_9"
→access="public" mangled="_ZN9unittests10test_suite3runEv" demangled=
→"unittests::test_suite::run()" location="f0:39" file="f0" line="39" extern="1"/>
  <Method id="_22" name="get_results" returns="_43" const="1" context="_9" access=
→"public" mangled="_ZNK9unittests10test_suite11get_resultsEv" demangled=
→"unittests::test_suite::get_results() const" location="f0:41" file="f0" line="41"
→endl="42" inline="1"/>
  <Enumeration id="_23" name="status" context="_11" access="public" location="f0:14"
→file="f0" line="14" artificial="1" size="32" align="32">
    <EnumValue name="ok" init="0"/>
    <EnumValue name="fail" init="1"/>
    <EnumValue name="error" init="2"/>
  </Enumeration>
  <Destructor id="_24" name="test_results" artificial="1" throw="" context="_11"
→access="public" mangled="_ZN9unittests12test_resultsD1Ev *INTERNAL* " demangled=
→"unittests::test_results::~~test_results()" location="f0:12" file="f0" line="12"
→endl="12" inline="1">
  </Destructor>
  <OperatorMethod id="_25" name="" returns="_44" artificial="1" throw="" context="_11
→" access="public" mangled="_ZN9unittests12test_resultsaSERKS0_" demangled=
→"unittests::test_results::operator=(unittests::test_results const&)" location=
→"f0:12" file="f0" line="12" endl="12" inline="1">

```

```

    <Argument type="_43" location="f0:12" file="f0" line="12"/>
  </OperatorMethod>
  <Constructor id="_26" name="test_results" artificial="1" throw="" context="_11"
↳access="public" mangled="_ZN9unittests12test_resultsC1ERKS0_ *INTERNAL* " demangled=
↳"unittests::test_results::test_results(unittests::test_results const&)"
↳location="f0:12" file="f0" line="12" inline="1">
    <Argument type="_43" location="f0:12" file="f0" line="12"/>
  </Constructor>
  <Constructor id="_27" name="test_results" artificial="1" throw="" context="_11"
↳access="public" mangled="_ZN9unittests12test_resultsC1Ev *INTERNAL* " demangled=
↳"unittests::test_results::test_results()" location="f0:12" file="f0" line="12"
↳inline="1"/>
  <Method id="_28" name="update" returns="_42" context="_11" access="public" mangled=
↳"_ZN9unittests12test_results6updateEPKcNS0_6statusE" demangled="unittests::test_
↳results::update(char const*, unittests::test_results::status)" location="f0:16"
↳file="f0" line="16" extern="1">
    <Argument name="test_name" type="_40" location="f0:16" file="f0" line="16"/>
    <Argument name="result" type="_23" location="f0:16" file="f0" line="16"/>
  </Method>
  <Field id="_29" name="m_name" type="_40" offset="32" context="_12" access="private"
↳location="f0:30" file="f0" line="30"/>
  <Destructor id="_30" name="test_case" artificial="1" throw="" context="_12" access=
↳"public" mangled="_ZN9unittests9test_caseD1Ev *INTERNAL* " demangled=
↳"unittests::test_case::~~test_case()" location="f0:19" file="f0" line="19" endl=
↳"19" inline="1">
  </Destructor>
  <OperatorMethod id="_31" name="" returns="_45" artificial="1" throw="" context="_12
↳" access="public" mangled="_ZN9unittests9test_caseaSERKS0_" demangled=
↳"unittests::test_case::operator=(unittests::test_case const&)" location="f0:19"
↳file="f0" line="19" endl="19" inline="1">
    <Argument type="_46" location="f0:19" file="f0" line="19"/>
  </OperatorMethod>
  <Constructor id="_32" name="test_case" artificial="1" throw="" context="_12" access=
↳"public" mangled="_ZN9unittests9test_caseC1ERKS0_ *INTERNAL* " demangled=
↳"unittests::test_case::test_case(unittests::test_case const&)" location="f0:19"
↳file="f0" line="19" endl="19" inline="1">
    <Argument type="_46" location="f0:19" file="f0" line="19"/>
  </Constructor>
  <Constructor id="_33" name="test_case" context="_12" access="public" mangled="_
↳ZN9unittests9test_caseC1EPKc *INTERNAL* " demangled="unittests::test_case::test_
↳case(char const*)" location="f0:21" file="f0" line="21" extern="1">
    <Argument name="test_case_name" type="_40" location="f0:21" file="f0" line="21"/>
  </Constructor>
  <Method id="_34" name="set_up" returns="_42" virtual="1" overrides="" context="_12"
↳access="public" mangled="_ZN9unittests9test_case6set_upEv" demangled=
↳"unittests::test_case::set_up()" location="f0:23" file="f0" line="23" inline="1"/>
  <Method id="_35" name="tear_down" returns="_42" virtual="1" overrides="" context="_
↳12" access="public" mangled="_ZN9unittests9test_case9tear_downEv" demangled=
↳"unittests::test_case::tear_down()" location="f0:25" file="f0" line="25" inline="1"/
↳>
  <Method id="_36" name="run" returns="_42" virtual="1" pure_virtual="1" overrides=""
↳context="_12" access="public" mangled="_ZN9unittests9test_case3runEv" demangled=
↳"unittests::test_case::run()" location="f0:27" file="f0" line="27" extern="1"/>
  <PointerType id="_37" type="_10" size="32" align="32"/>
  <ReferenceType id="_38" type="_9" size="32" align="32"/>
  <ReferenceType id="_39" type="_9c" size="32" align="32"/>
  <PointerType id="_40" type="_48c" size="32" align="32"/>
  <ReferenceType id="_41" type="_10c" size="32" align="32"/>

```

```

<FundamentalType id="_42" name="void" align="8"/>
<ReferenceType id="_43" type="_11c" size="32" align="32"/>
<ReferenceType id="_44" type="_11" size="32" align="32"/>
<ReferenceType id="_45" type="_12" size="32" align="32"/>
<ReferenceType id="_46" type="_12c" size="32" align="32"/>
<FundamentalType id="_48" name="char" size="8" align="8"/>
<CvQualifiedType id="_48c" type="_48" const="1"/>
<CvQualifiedType id="_11c" type="_11" const="1"/>
<CvQualifiedType id="_12c" type="_12" const="1"/>
<CvQualifiedType id="_10c" type="_10" const="1"/>
<CvQualifiedType id="_9c" type="_9" const="1"/>
<File id="f0" name="example.hpp"/>
<File id="f1" name="&lt;built-in&gt;"/>
</GCC_XML>

```

## Python API usage example

```

import os
import sys

# Find out the file location within the sources tree
this_module_dir_path = os.path.abspath(
    os.path.dirname(sys.modules[__name__].__file__))
# Add pygccxml package to Python path
sys.path.append(os.path.join(this_module_dir_path, '..', '..'))

from pygccxml import parser # nopep8
from pygccxml import declarations # nopep8
from pygccxml import utils # nopep8

# Find out the xml generator (gccxml or castxml)
generator_path, generator_name = utils.find_xml_generator()

# Configure the xml generator
config = parser.xml_generator_configuration_t(
    xml_generator_path=generator_path,
    xml_generator=generator_name,
    compiler="gcc")

# Parsing source file
decls = parser.parse([this_module_dir_path + '/example.hpp'], config)
global_ns = declarations.get_global_namespace(decls)

# Get object that describes unittests namespace
unittests = global_ns.namespace('unittests')

print("unittests" declarations: \n')
declarations.print_declarations(unittests)

# Print all base and derived class names
for class_ in unittests.classes():
    print('class "%s" hierarchy information:' % class_.name)
    print('\tbase classes : ', repr([
        base.related_class.name for base in class_.bases]))

```

```
print('\tderived classes: ', repr([
    derive.related_class.name for derive in class_.derived]))
print('\n')

# Pygccxml has very powerfull query api:

# Select multiple declarations
run_functions = unittests.member_functions('run')
print('the namespace contains %d "run" member functions' % len(run_functions))
print('they are: ')
for f in run_functions:
    print('\t' + declarations.full_name(f))

# Select single declaration - all next statements will return same object
# vector< unittests::test_case* >

# You can select the class using "full" name
test_container_1 = global_ns.class_('::unittests::test_suite')
# You can define your own "match" criteria
test_container_2 = global_ns.class_(lambda decl: 'suite' in decl.name)

is_same_object = test_container_1 is test_container_2
print(
    "Does all test_container_* refer to the same object? " +
    str(is_same_object))
```

## Output

```
>e:\Python26\pythonw.exe -u "example.py"
D:\dev\language-binding\sources\pygccxml_dev\docs\example\...\
↪pygccxml\parser\declarations_cache.py:8: DeprecationWarning: the md5 module is_
↪deprecated; use hashlib instead
import md5
INFO Parsing source file "example.hpp" ...
INFO gccxml cmd: "D:\dev\language-binding\sources\gccxml_bin\v09\win32\bin\gccxml.exe
↪" -I"." "example.hpp" -fxml="e:\docume~1\romany\locals~1\temp\tmpewcrem.xml" --
↪gccxml-compiler msvc71"
INFO GCCXML version - 0.9( 1.127 )
"unittests" declarations:

namespace_t: 'unittests'

    artificial: 'False'

    demangled: unittests

    mangled: _Z9unittests

    class_t: 'test_suite'

        location: [D:\dev\language-binding\sources\pygccxml_dev\docs\example\example.
↪hpp]:35

        artificial: '1'

        demangled: unittests::test_suite
```

```

mangled: N9unittests10test_suiteE

class type: 'struct'

size: 16

align: 4

base classes:

    class: '::unittests::test_case'

        access type: 'public'

        virtual inheritance: 'False'

public:

    destructor_t: '~test_suite'

        location: [D:\dev\language-binding\sources\pygccxml_
↪dev\docs\example\example.hpp]:35

        artificial: '1'

        demangled: unittests::test_suite::~~test_suite()

        mangled: _ZN9unittest

    member_operator_t: 'operator='

        location: [D:\dev\language-binding\sources\pygccxml_
↪dev\docs\example\example.hpp]:35

        artificial: '1'

        demangled: unittests::test_suite::operator=(unittests::test_suite const&)

        mangled: _ZN9unittests10test_suiteaSERKS0_

        is extern: False

        return type: ::unittests::test_suite &

        arguments type: ::unittests::test_suite const & arg0

        calling convention: __thiscall__

        virtual: not virtual

        is const: False

        is static: False

    constructor_t: 'test_suite'

        location: [D:\dev\language-binding\sources\pygccxml_
↪dev\docs\example\example.hpp]:35

```

```
artificial: '1'

demangled: unittests::test_suite::test_suite(unittests::test_suite const&)
mangled: _ZN9unittest

is extern: False

return type: None

arguments type: ::unittests::test_suite const & arg0

calling convention: __thiscall__

virtual: not virtual

is const: False

is static: False

copy constructor: True

constructor_t: 'test_suite'

location: [D:\dev\language-binding\sources\pygccxml_
↳dev\docs\example\example.hpp]:37

artificial: 'False'

demangled: unittests::test_suite::test_suite(char const*, unittests::test_
↳container const&)

mangled: _ZN9unittest

is extern: 1

return type: None

arguments type: char const * name, ::unittests::test_container const &_
↳tests

calling convention: __thiscall__

virtual: not virtual

is const: False

is static: False

copy constructor: False

member_function_t: 'run'

location: [D:\dev\language-binding\sources\pygccxml_
↳dev\docs\example\example.hpp]:39

artificial: 'False'
```



```

    demangled: unittests::test_suite::run()
    mangled: _ZN9unittests10test_suite3runEv
    is extern: 1
    return type: void
    arguments type:
    calling convention: __thiscall__
    virtual: virtual
    is const: False
    is static: False
    member_function_t: 'get_results'

    location: [D:\dev\language-binding\sources\pygccxml_
→dev\docs\example\example.hpp]:41

    artificial: 'False'
    demangled: unittests::test_suite::get_results() const
    mangled: _ZNK9unittests10test_suite11get_resultsEv
    is extern: False
    return type: ::unittests::test_results const &
    arguments type:
    calling convention: __thiscall__
    virtual: not virtual
    is const: 1
    is static: False
    protected:
    private:
    variable_t: 'm_tests'

    location: [D:\dev\language-binding\sources\pygccxml_
→dev\docs\example\example.hpp]:45

    artificial: 'False'

    type: ::unittests::test_container *
    value: None

```

```
        size: 4

        align: 4

        offset: 8

    variable_t: 'm_results'

        location: [D:\dev\language-binding\sources\pygccxml_
↪dev\docs\example\example.hpp]:46

        artificial: 'False'

        type: ::unittests::test_results

        value: None

        size: 1

        align: 1

        offset: 12

    class_declaration_t: 'test_container'

        location: [D:\dev\language-binding\sources\pygccxml_dev\docs\example\example.
↪hpp]:33

        artificial: '1'

        demangled: unittests::test_container

        mangled: N9unittests14test_containerE

    class_t: 'test_results'

        location: [D:\dev\language-binding\sources\pygccxml_dev\docs\example\example.
↪hpp]:12

        artificial: '1'

        demangled: unittests::test_results

        mangled: N9unittests12test_resultsE

        class type: 'struct'

        size: 1

        align: 1

        public:

        enumeration_t: 'status'
```

```

        location: [D:\dev\language-binding\sources\pygccxml_
↪dev\docs\example\example.hpp]:14

        artificial: '1'

        values:

            ok : 0

            fail : 1

            error : 2

        destructor_t: '~test_results'

        location: [D:\dev\language-binding\sources\pygccxml_
↪dev\docs\example\example.hpp]:12

        artificial: '1'

        demangled: unittests::test_results::~~test_results()

        mangled: _ZN9unittest

        member_operator_t: 'operator='

        location: [D:\dev\language-binding\sources\pygccxml_
↪dev\docs\example\example.hpp]:12

        artificial: '1'

        demangled: unittests::test_results::operator=(unittests::test_results_
↪const&)

        mangled: _ZN9unittests12test_resultsaSERKS0_

        is extern: False

        return type: ::unittests::test_results &

        arguments type: ::unittests::test_results const & arg0

        calling convention: __thiscall__

        virtual: not virtual

        is const: False

        is static: False

        constructor_t: 'test_results'

        location: [D:\dev\language-binding\sources\pygccxml_
↪dev\docs\example\example.hpp]:12

```

```
        artificial: '1'

        demangled: unittests::test_results::test_results(unittests::test_results_
↳const&)

        mangled: _ZN9unittest

        is extern: False

        return type: None

        arguments type: ::unittests::test_results const & arg0

        calling convention: __thiscall__

        virtual: not virtual

        is const: False

        is static: False

        copy constructor: True

    constructor_t: 'test_results'

        location: [D:\dev\language-binding\sources\pygccxml_
↳dev\docs\example\example.hpp]:12

        artificial: '1'

        demangled: unittests::test_results::test_results()

        mangled: _ZN9unittest

        is extern: False

        return type: None

        arguments type:

        calling convention: __thiscall__

        virtual: not virtual

        is const: False

        is static: False

        copy constructor: False

    member_function_t: 'update'

        location: [D:\dev\language-binding\sources\pygccxml_
↳dev\docs\example\example.hpp]:16

        artificial: 'False'

        demangled: unittests::test_results::update(char const*, unittests::test_
↳results::status)
```

```

    mangled: _ZN9unittests12test_results6updateEPKcNS0_6statusE
    is extern: 1
    return type: void
    arguments type: char const * test_name, ::unittests::test_results::status_
↪result
    calling convention: __thiscall__
    virtual: not virtual
    is const: False
    is static: False
    protected:
    private:
    class_t: 'test_case'
    location: [D:\dev\language-binding\sources\pygccxml_dev\docs\example\example.
↪hpp]:19
    artificial: '1'
    demangled: unittests::test_case
    mangled: N9unittests9test_caseE
    class type: 'struct'
    size: 8
    align: 4
    derived classes:
        class: '::unittests::test_suite'
            access type: 'public'
            virtual inheritance: 'False'
    public:
    destructor_t: '~test_case'
        location: [D:\dev\language-binding\sources\pygccxml_
↪dev\docs\example\example.hpp]:19
        artificial: '1'
        demangled: unittests::test_case::~~test_case()

```

```
    mangled: _ZN9unittest

    member_operator_t: 'operator='

        location: [D:\dev\language-binding\sources\pygccxml_
↪dev\docs\example\example.hpp]:19

        artificial: '1'

        demangled: unittests::test_case::operator=(unittests::test_case const&)

        mangled: _ZN9unittests9test_caseaSERKS0_

        is extern: False

        return type: ::unittests::test_case &

        arguments type: ::unittests::test_case const & arg0

        calling convention: __thiscall__

        virtual: not virtual

        is const: False

        is static: False

    constructor_t: 'test_case'

        location: [D:\dev\language-binding\sources\pygccxml_
↪dev\docs\example\example.hpp]:19

        artificial: '1'

        demangled: unittests::test_case::test_case(unittests::test_case const&)

        mangled: _ZN9unittest

        is extern: False

        return type: None

        arguments type: ::unittests::test_case const & arg0

        calling convention: __thiscall__

        virtual: not virtual

        is const: False

        is static: False

        copy constructor: True

    constructor_t: 'test_case'

        location: [D:\dev\language-binding\sources\pygccxml_
↪dev\docs\example\example.hpp]:21
```

```

        artificial: 'False'

        demangled: unittests::test_case::test_case(char const*)

        mangled: _ZN9unittest

        is extern: 1

        return type: None

        arguments type: char const * test_case_name

        calling convention: __thiscall__

        virtual: not virtual

        is const: False

        is static: False

        copy constructor: False

        member_function_t: 'set_up'

        location: [D:\dev\language-binding\sources\pygccxml_
→dev\docs\example\example.hpp]:23

        artificial: 'False'

        demangled: unittests::test_case::set_up()

        mangled: _ZN9unittests9test_case6set_upEv

        is extern: False

        return type: void

        arguments type:

        calling convention: __thiscall__

        virtual: virtual

        is const: False

        is static: False

        member_function_t: 'tear_down'

        location: [D:\dev\language-binding\sources\pygccxml_
→dev\docs\example\example.hpp]:25

        artificial: 'False'

        demangled: unittests::test_case::tear_down()

        mangled: _ZN9unittests9test_case9tear_downEv

```

```
is extern: False

return type: void

arguments type:

calling convention: __thiscall__

virtual: virtual

is const: False

is static: False

member_function_t: 'run'

location: [D:\dev\language-binding\sources\pygccxml_
→dev\docs\example\example.hpp]:27

artificial: 'False'

demangled: unittests::test_case::run()

mangled: _ZN9unittests9test_case3runEv

is extern: 1

return type: void

arguments type:

calling convention: __thiscall__

virtual: pure virtual

is const: False

is static: False

protected:

private:

variable_t: 'm_name'

location: [D:\dev\language-binding\sources\pygccxml_
→dev\docs\example\example.hpp]:30

artificial: 'False'

type: char const *

value: None

size: 4

align: 4
```



```

        offset: 4

class "test_suite" hierarchy information:
    base classes    : ['test_case']
    derived classes: []

class "test_results" hierarchy information:
    base classes    : []
    derived classes: []

class "test_case" hierarchy information:
    base classes    : []
    derived classes: ['test_suite']

the namespace contains 2 "run" member functions
they are:
    ::unittests::test_suite::run
    ::unittests::test_case::run
Does all test_container_* refer to the same object?  True
>Exit code: 0

```

## FAQ

### GCCXML vs CastXML

GCCXML has been superseded by CastXML. It is highly recommended to use CastXML. GCCXML support will be removed from Pygccxml in version 2.0.

### C++ and C code support

Pygccxml supports C++98, as CastXML and GCCXML only output declarations from the C++98 subset. Of course, newer versions of C++ can be parsed (the tests currently all pass with C++11 and C++14), but not all new features from these language definitions can be used.

C code support has been reported to work. As C is similar to C++, this makes sense. Some discrepancies may be present.

Still, parsing C code is not officially supported by pygccxml, as it falls out of scope of this project. Of course, if some volunteer wants to work on this, submissions would be accepted.

### Function and method bodies

Pygccxml does not allow to fetch declarations defined in function or method bodies. For example the following a variable will not appear in the declarations tree:

```

int f() {
    int a = 3;

```

```
        return a;
    }
```

Neither GCCXML or CastXML currently support this feature. CastXML could probably be extended for this later, as pygccxml.

## Performance

pygccxml is being regularly optimised for performance, but it still may be slow in certain cases.

Before all, it is highly recommended to benchmark your application if performance is important to you. There are multiple tools out there for benchmarking python applications. We currently are using the following two command lines / tools:

```
python -m cProfile -o profile_data.pyprof script_to_profile.py
pyprof2calltree -i profile_data.pyprof -k
```

Of course optimising pygccxml alone will not help in all cases. The bottlenecks can also be in the code calling pygccxml, to make sure to benchmark the whole process. Any help on the performance side is also welcome.

Some things you may try (in order of priority):

1. You might want to consider making the declaration tree as small as possible and only store those declarations that somehow have an influence on the bindings. Ideally, this is done as early as possible and luckily castxml and gccxml provide an option that allows you to reduce the number of declarations that need to be parsed.

You can specify one or more declarations using the `-fxml-start` (gccxml) or `-castxml-start` (castxml) options when running the xml generator. For example, if you specify the name of a particular class, only this class and all its members will get written. Ideally, your project should already use a dedicated namespace, which you can then use as a starting point. All declarations stemming from system headers will be ignored (except for those declarations that are actually used within your library).

In the pygccxml package you can set the value for these flags by using the `start_with_declarations` attribute of the `pygccxml.parser.config_t` object that you are passing to the parser.

2. You can pass the following flag to the `read_files` method:

```
compilation_mode=pygccxml.parser.COMPIRATION_MODE.ALL_AT_ONCE
```

3. If you want to cache the declarations tree, there is a caching mechanism provided by pygccxml. You will find an example of this mechanism in the examples section.

## Flags

### castxml\_epic\_version

The ``castxml_epic_version`` can be set to 1 to benefit from new castxml and pygccxml features. To be able to use this, you will need the latest castxml version.

Currently this adds the support for elaborated type specifiers.

### \_\_va\_list\_tag and other hidden declarations (f1)

When parsing with CastXML, the XML tree can contain declarations named `__va_list_tag`. If the compiler is llvm 3.9, `__NSConstantString_tag` and `__NSConstantString` declarations may also be present.

These declarations are internal declarations, coming from the std c++ library headers you include, and are often not needed. They are for example polluting the declarations tree when running pyplusplus.

By default, pygccxml will ignore these declarations. To still read these declarations from the xml file, a config flag can be set (`config.flags = ["f1"]`), or a flag can be passed as argument the config setup (`flags=["f1"]`).

## `__thiscall__` in attributes (f2)

Attributes defined as `__thiscall__` are now ignored (tested with VS 2013). The `__thiscall__` in some attributes will be removed too. If you still want to have access to these attributes, you can use the `config.flags = ["f2"]` option.

## API

### pygccxml package

Python CastXML or GCC-XML front end.

This package provides functionality to extract and inspect declarations from C/C++ header files. This is accomplished by invoking an external tool like CastXML or GCC-XML, which parses a header file and dumps the declarations as a XML file. This XML file is then read by pygccxml and the contents are made available as appropriate Python objects.

To parse a set of C/C++ header files you use the `parse` function in the `:mod:parser` sub package which returns a tree that contains all declarations found in the header files. The root of the tree represents the main namespace `::` and the children nodes represent the namespace contents such as other namespaces, classes, functions, etc. Each node in the tree is an object of a type derived from the `declaration_t` class. An inner node is always either a namespace `declarations.namespace_t` or a class `declarations.class_t`, which are both derived from `declarations.scopeddef_t` class. Everything else (free functions, member functions, enumerations, variables, etc.) are always a leaf. You will find all those declaration classes in the `:mod:declarations` sub-package.

### Subpackages

#### pygccxml.declarations package

Contains classes that describe different C++ declarations

##### **access\_type\_matcher**

see `access_type_matcher_t` for documentation

alias of `access_type_matcher_t`

##### **and\_matcher**

see `and_matcher_t` for documentation

alias of `and_matcher_t`

##### **calldef\_matcher**

see `calldef_matcher_t` for documentation

alias of `calldef_matcher_t`

##### **custom\_matcher**

see `custom_matcher_t` for documentation

alias of `custom_matcher_t`

**declaration\_matcher**

see `declaration_matcher_t` for documentation

alias of `declaration_matcher_t`

**namespace\_matcher**

see `namespace_matcher_t` for documentation

alias of `namespace_matcher_t`

**not\_matcher**

see `not_matcher_t` for documentation

alias of `not_matcher_t`

**operator\_matcher**

see `operator_matcher_t` for documentation

alias of `operator_matcher_t`

**or\_matcher**

see `or_matcher_t` for documentation

alias of `or_matcher_t`

**regex\_matcher**

see `regex_matcher_t` for documentation

alias of `regex_matcher_t`

**variable\_matcher**

see `variable_matcher_t` for documentation

alias of `variable_matcher_t`

**virtuality\_type\_matcher**

see `virtuality_type_matcher_t` for documentation

alias of `virtuality_type_matcher_t`

## Submodules

### pygccxml.declarations.algorithm module

Define few unrelated algorithms that work on declarations.

**apply\_visitor** (*visitor*, *decl\_inst*)

Applies a visitor on declaration instance.

**Parameters** **visitor** (`type_visitor_t` or `decl_visitor_t`) – instance

**class** **match\_declaration\_t** (*decl\_type=None*, *name=None*, *fullname=None*, *parent=None*)

Bases: `object`

Helper class for different search algorithms.

**This class will help developer to match declaration by:**

- **declaration type**, for example `class_t` or `operator_t`.
- declaration name
- declaration full name
- reference to parent declaration

**does\_match\_exist** (*inst*)

Returns True if inst does match one of specified criteria.

**Parameters** *inst* (declaration\_t) – declaration instance

**Return type** bool

### pygccxml.declarations.algorithms\_cache module

Defines classes that will keep results of different calculations.

**class** **declaration\_algs\_cache\_t**

Bases: object

**access\_type**

**cmp\_data**

Data used for comparison between declarations.

**container\_element\_type**

**container\_key\_type**

**container\_traits**

**declaration\_path**

**demangled\_name**

**disable()**

**enable()**

**enabled**

**full\_name**

**full\_partial\_name**

**normalized\_full\_name\_false**

**normalized\_full\_name\_true**

**normalized\_name**

**normalized\_partial\_name**

**partial\_declaration\_path**

**reset()**

**reset\_access\_type()**

**reset\_name\_based()**

**class** **type\_algs\_cache\_t**

Bases: object

**decl\_string**

**static disable()**

**static enable()**

**enabled = True**

**partial\_decl\_string**

**remove\_alias**

**reset** ()

## pygccxml.declarations.byte\_info module

**class byte\_info**

Bases: object

This class stores information about the byte size and byte align values from a declaration/type.

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

## pygccxml.declarations.call\_invocation module

Free function invocation parser

The parser is able to extract function name and list of arguments from a function invocation statement. For example, for the following code

```
do_something( x1, x2, x3 )
```

the parser will extract - function name - *do\_something* - argument names - [ *x1*, *x2*, *x3* ]

**args** (*declaration\_string*)

Returns list of function arguments

**Return type** [str]

**find\_args** (*text*, *start=None*)

Finds arguments within function invocation.

**Return type** [ arguments ] or :data:NOT\_FOUND if arguments could not be found.

**is\_call\_invocation** (*declaration\_string*)

Returns True if *declaration\_string* is a function invocation.

**Parameters** **declaration\_string** (*str*) – string that should be checked for pattern.

**Return type** bool

**join** (*name\_*, *args\_*, *arg\_separator=None*)

Returns name( argument\_1, argument\_2, ..., argument\_n ).

**name** (*declaration\_string*)

Returns the name of a function.

**Return type** str

**split** (*declaration\_string*)  
Returns (name, [arguments] )

**split\_recursive** (*declaration\_string*)  
Returns [(name, [arguments])].

## pygccxml.declarations.calldf module

defines classes, that describes “callable” declarations

This modules contains definition for next C++ declarations:

- **operator**
  - member
  - free
- **function**
  - member
  - free
- constructor
- destructor

**class argument\_t** (*name='', decl\_type=None, default\_value=None, attributes=None*)

Bases: `object`

class, that describes argument of “callable” declaration

**attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))` @type: str

**clone** (*\*\*keywd*)

constructs new `argument_t` instance

**return argument\_t**( name=keywd.get('name', self.name), decl\_type=keywd.get('decl\_type', self.decl\_type), default\_value=keywd.get('default\_value', self.default\_value), attributes=keywd.get('attributes', self.attributes))

**decl\_type**

**default\_value**

Argument’s default value or None. @type: str

**ellipsis**

bool, if True argument represents ellipsis ( "..." ) in function definition

**name**

Argument name. @type: str

**class calldf\_t** (*name='', arguments=None, exceptions=None, return\_type=None, has\_extern=False, does\_throw=True, mangled=None*)

Bases: `pygccxml.declarations.declaration.declaration_t`

base class for all “callable” declarations

**argument\_types**

list of all argument types

**arguments**

The argument list. @type: list of `argument_t`

**attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

**cache**

Implementation detail.

Reference to instance of `algorithms_cache_t` class.

**calling\_convention**

function calling convention. See `:class:CALLING_CONVENTION_TYPES` class for possible values

**create\_decl\_string** (*with\_defaults=True*)**decl\_string**

Declaration full name.

**decorated\_name**

Unique declaration name extracted from a binary file ( `.map`, `.dll`, `.so`, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**demangled\_name**

returns function demangled name. It can help you to deal with function template instantiations

**does\_throw**

If False, than function does not throw any exception. In this case, function was declared with empty throw statement.

**exceptions**

The list of exceptions. @type: list of `declaration_t`

**get\_mangled\_name** ()**guess\_calling\_convention** ()

This function should be overridden in the derived classes and return more-or-less successfull guess about calling convention

**has\_ellipsis****has\_extern**

Was this callable declared as “extern”? @type: bool

**has\_inline**

Was this callable declared with “inline” specifier @type: bool

**i\_depend\_on\_them** (*recursive=True*)**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: `location_t`

**mangled**

Unique declaration name generated by the compiler.



**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**optional\_args**

list of all optional arguments, the arguments that have default value

**overloads**

A list of overloaded “callables” (i.e. other callables with the same name within the same scope.

@type: list of *calldef\_t*

**parent**

Reference to parent declaration.

@type: *declaration\_t*

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**required\_args**

list of all required arguments

**return\_type**

The type of the return value of the “callable” or None (constructors). @type: *type\_t*

**top\_parent**

Reference to top parent declaration.

@type: *declaration\_t*

## pygccxml.declarations.calldef\_members module

**class** *casting\_operator\_t* (\*args, \*\*keywords)

Bases: *pygccxml.declarations.calldef\_members.member\_calldef\_t*, *pygccxml.declarations.calldef\_members.operator\_t*

describes casting operator declaration

**OPERATOR\_WORD\_LEN** = 8

**access\_type**

Return the access type of the member (as defined by the string constants in the class :class:ACCESS\_TYPES. @type: str

**argument\_types**

list of all argument types

**arguments**

The argument list. @type: list of *argument\_t*

**attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

**cache**

Implementation detail.

Reference to instance of `algorithms_cache_t` class.

**calling\_convention**

function calling convention. See `:class:CALLING_CONVENTION_TYPES` class for possible values

**create\_decl\_string** (*with\_defaults=True*)**decl\_string**

Declaration full name.

**decorated\_name**

Unique declaration name extracted from a binary file ( `.map`, `.dll`, `.so`, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**demangled\_name**

returns function demangled name. It can help you to deal with function template instantiations

**does\_throw**

If False, than function does not throw any exception. In this case, function was declared with empty throw statement.

**exceptions**

The list of exceptions. @type: list of `declaration_t`

**function\_type** ()

returns function type. See `type_t` hierarchy

**get\_mangled\_name** ()**guess\_calling\_convention** ()**has\_const**

describes, whether “callable” has const modifier or not

**has\_ellipsis****has\_extern**

Was this callable declared as “extern”? @type: bool

**has\_inline**

Was this callable declared with “inline” specifier @type: bool

**has\_static**

describes, whether “callable” has static modifier or not

**i\_depend\_on\_them** (*recursive=True*)**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: `location_t`

**mangled**

Unique declaration name generated by the compiler.

**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**optional\_args**

list of all optional arguments, the arguments that have default value

**overloads**

A list of overloaded “callables” (i.e. other callables with the same name within the same scope.

@type: list of `calldef_t`

**parent**

Reference to parent declaration.

@type: `declaration_t`

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**required\_args**

list of all required arguments

**return\_type**

The type of the return value of the “callable” or None (constructors). @type: `type_t`

**symbol**

*operator's symbol. For example – operator+, symbol is equal to '+'*

**top\_parent**

Reference to top parent declaration.

@type: `declaration_t`

**virtuality**

Describes the “virtuality” of the member (as defined by the string constants in the class :class:VIRTUALITY\_TYPES). @type: str

**class constructor\_t** (\*args, \*\*keywords)

Bases: `pygccxml.declarations.calldef_members.member_calldef_t`

describes constructor declaration

**access\_type**

Return the access type of the member (as defined by the string constants in the class :class:ACCESS\_TYPES). @type: str

**argument\_types**

list of all argument types

**arguments**

The argument list. @type: list of `argument_t`

**attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

**cache**

Implementation detail.

Reference to instance of `algorithms_cache_t` class.

**calling\_convention**

function calling convention. See `:class:CALLING_CONVENTION_TYPES` class for possible values

**create\_decl\_string** (*with\_defaults=True*)**decl\_string**

Declaration full name.

**decorated\_name**

Unique declaration name extracted from a binary file ( `.map`, `.dll`, `.so`, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**demangled\_name**

returns function demangled name. It can help you to deal with function template instantiations

**does\_throw**

If False, than function does not throw any exception. In this case, function was declared with empty throw statement.

**exceptions**

The list of exceptions. @type: list of `declaration_t`

**explicit**

True, if constructor has “explicit” keyword, False otherwise @type: bool

**function\_type** ()

returns function type. See `type_t` hierarchy

**get\_mangled\_name** ()**guess\_calling\_convention** ()**has\_const**

describes, whether “callable” has const modifier or not

**has\_ellipsis****has\_extern**

Was this callable declared as “extern”? @type: bool

**has\_inline**

Was this callable declared with “inline” specifier @type: bool

**has\_static**

describes, whether “callable” has static modifier or not

**i\_depend\_on\_them** (*recursive=True*)

**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: location\_t

**mangled**

Unique declaration name generated by the compiler.

**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**optional\_args**

list of all optional arguments, the arguments that have default value

**overloads**

A list of overloaded “callables” (i.e. other callables with the same name within the same scope.

@type: list of calldef\_t

**parent**

Reference to parent declaration.

@type: declaration\_t

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**required\_args**

list of all required arguments

**return\_type**

The type of the return value of the “callable” or None (constructors). @type: type\_t

**top\_parent**

Reference to top parent declaration.

@type: declaration\_t

**virtuality**

Describes the “virtuality” of the member (as defined by the string constants in the class :class:VIRTUALITY\_TYPES). @type: str

**class destructor\_t** (\*args, \*\*keywords)

Bases: `pygccxml.declarations.calldef_members.member_calldef_t`

describes deconstructor declaration

**access\_type**

Return the access type of the member (as defined by the string constants in the class :class:ACCESS\_TYPES). @type: str

**argument\_types**

list of all argument types

**arguments**

The argument list. @type: list of `argument_t`

**attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

**cache**

Implementation detail.

Reference to instance of `algorithms_cache_t` class.

**calling\_convention**

function calling convention. See :class:CALLING\_CONVENTION\_TYPES class for possible values

**create\_decl\_string** (*with\_defaults=True*)**decl\_string**

Declaration full name.

**decorated\_name**

Unique declaration name extracted from a binary file ( `.map`, `.dll`, `.so`, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**demangled\_name**

returns function demangled name. It can help you to deal with function template instantiations

**does\_throw**

If False, than function does not throw any exception. In this case, function was declared with empty throw statement.

**exceptions**

The list of exceptions. @type: list of `declaration_t`

**function\_type** ()

returns function type. See `type_t` hierarchy

**get\_mangled\_name** ()**guess\_calling\_convention** ()**has\_const**

describes, whether “callable” has const modifier or not

**has\_ellipsis****has\_extern**

Was this callable declared as “extern”? @type: bool

**has\_inline**

Was this callable declared with “inline” specifier @type: bool

**has\_static**

describes, whether “callable” has static modifier or not

**i\_depend\_on\_them** (*recursive=True*)

**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: location\_t

**mangled**

Unique declaration name generated by the compiler.

**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**optional\_args**

list of all optional arguments, the arguments that have default value

**overloads**

A list of overloaded “callables” (i.e. other callables with the same name within the same scope.

@type: list of calldef\_t

**parent**

Reference to parent declaration.

@type: declaration\_t

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**required\_args**

list of all required arguments

**return\_type**

The type of the return value of the “callable” or None (constructors). @type: type\_t

**top\_parent**

Reference to top parent declaration.

@type: declaration\_t

**virtuality**

Describes the “virtuality” of the member (as defined by the string constants in the class :class:VIRTUALITY\_TYPES). @type: str

**class member\_calldef\_t** (*virtuality=None, has\_const=None, has\_static=None, \*args, \*\*keywords*)

Bases: [pygccxml.declarations.calldef.calldef\\_t](#)

base class for “callable” declarations that defined within C++ class or struct

**access\_type**

Return the access type of the member (as defined by the string constants in the class :class:ACCESS\_TYPES). @type: str

**argument\_types**

list of all argument types

**arguments**

The argument list. @type: list of `argument_t`

**attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

**cache**

Implementation detail.

Reference to instance of `algorithms_cache_t` class.

**calling\_convention**

function calling convention. See :class:CALLING\_CONVENTION\_TYPES class for possible values

**create\_decl\_string** (*with\_defaults=True*)**decl\_string**

Declaration full name.

**decorated\_name**

Unique declaration name extracted from a binary file ( `.map`, `.dll`, `.so`, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**demangled\_name**

returns function demangled name. It can help you to deal with function template instantiations

**does\_throw**

If False, than function does not throw any exception. In this case, function was declared with empty throw statement.

**exceptions**

The list of exceptions. @type: list of `declaration_t`

**function\_type** ()

returns function type. See `type_t` hierarchy

**get\_mangled\_name** ()**guess\_calling\_convention** ()**has\_const**

describes, whether “callable” has `const` modifier or not

**has\_ellipsis****has\_extern**

Was this callable declared as “extern”? @type: bool

**has\_inline**

Was this callable declared with “inline” specifier @type: bool

**has\_static**

describes, whether “callable” has `static` modifier or not

**i\_depend\_on\_them** (*recursive=True*)



**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: location\_t

**mangled**

Unique declaration name generated by the compiler.

**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**optional\_args**

list of all optional arguments, the arguments that have default value

**overloads**

A list of overloaded “callables” (i.e. other callables with the same name within the same scope.

@type: list of calldef\_t

**parent**

Reference to parent declaration.

@type: declaration\_t

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**required\_args**

list of all required arguments

**return\_type**

The type of the return value of the “callable” or None (constructors). @type: type\_t

**top\_parent**

Reference to top parent declaration.

@type: declaration\_t

**virtuality**

Describes the “virtuality” of the member (as defined by the string constants in the class :class:VIRTUALITY\_TYPES). @type: str

**class member\_function\_t** (\*args, \*\*keywords)

Bases: `pygccxml.declarations.calldef_members.member_calldef_t`

describes member function declaration

**access\_type**

Return the access type of the member (as defined by the string constants in the class :class:ACCESS\_TYPES). @type: str

**argument\_types**

list of all argument types

**arguments**

The argument list. @type: list of `argument_t`

**attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

**cache**

Implementation detail.

Reference to instance of `algorithms_cache_t` class.

**calling\_convention**

function calling convention. See :class:CALLING\_CONVENTION\_TYPES class for possible values

**create\_decl\_string** (*with\_defaults=True*)**decl\_string**

Declaration full name.

**decorated\_name**

Unique declaration name extracted from a binary file ( `.map`, `.dll`, `.so`, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**demangled\_name**

returns function demangled name. It can help you to deal with function template instantiations

**does\_throw**

If False, than function does not throw any exception. In this case, function was declared with empty throw statement.

**exceptions**

The list of exceptions. @type: list of `declaration_t`

**function\_type** ()

returns function type. See `type_t` hierarchy

**get\_mangled\_name** ()**guess\_calling\_convention** ()**has\_const**

describes, whether “callable” has const modifier or not

**has\_ellipsis****has\_extern**

Was this callable declared as “extern”? @type: bool

**has\_inline**

Was this callable declared with “inline” specifier @type: bool

**has\_static**

describes, whether “callable” has static modifier or not

**i\_depend\_on\_them** (*recursive=True*)

**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: location\_t

**mangled**

Unique declaration name generated by the compiler.

**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**optional\_args**

list of all optional arguments, the arguments that have default value

**overloads**

A list of overloaded “callables” (i.e. other callables with the same name within the same scope.

@type: list of calldef\_t

**parent**

Reference to parent declaration.

@type: declaration\_t

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**required\_args**

list of all required arguments

**return\_type**

The type of the return value of the “callable” or None (constructors). @type: type\_t

**top\_parent**

Reference to top parent declaration.

@type: declaration\_t

**virtuality**

Describes the “virtuality” of the member (as defined by the string constants in the class :class:VIRTUALITY\_TYPES). @type: str

**class member\_operator\_t** (\*args, \*\*keywords)

Bases: `pygccxml.declarations.calldef_members.member_calldef_t`, `pygccxml.declarations.calldef_members.operator_t`

describes member operator declaration

**OPERATOR\_WORD\_LEN** = 8

**access\_type**

Return the access type of the member (as defined by the string constants in the class :class:ACCESS\_TYPES. @type: str

**argument\_types**

list of all argument types

**arguments**

The argument list. @type: list of argument\_t

**attributes**

GCCXML attributes, set using \_\_attribute\_\_((gccxml("...")))

@type: str

**cache**

Implementation detail.

Reference to instance of algorithms\_cache\_t class.

**calling\_convention**

function calling convention. See :class:CALLING\_CONVENTION\_TYPES class for possible values

**create\_decl\_string** (*with\_defaults=True*)**decl\_string**

Declaration full name.

**decorated\_name**

Unique declaration name extracted from a binary file ( .map, .dll, .so, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**demangled\_name**

returns function demangled name. It can help you to deal with function template instantiations

**does\_throw**

If False, than function does not throw any exception. In this case, function was declared with empty throw statement.

**exceptions**

The list of exceptions. @type: list of declaration\_t

**function\_type** ()

returns function type. See type\_t hierarchy

**get\_mangled\_name** ()**guess\_calling\_convention** ()**has\_const**

describes, whether “callable” has const modifier or not

**has\_ellipsis****has\_extern**

Was this callable declared as “extern”? @type: bool

**has\_inline**

Was this callable declared with “inline” specifier @type: bool

**has\_static**

describes, whether “callable” has static modifier or not

**i\_depend\_on\_them** (*recursive=True*)**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: location\_t

**mangled**

Unique declaration name generated by the compiler.

**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**optional\_args**

list of all optional arguments, the arguments that have default value

**overloads**

A list of overloaded “callables” (i.e. other callables with the same name within the same scope.

@type: list of calldef\_t

**parent**

Reference to parent declaration.

@type: declaration\_t

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**required\_args**

list of all required arguments

**return\_type**

The type of the return value of the “callable” or None (constructors). @type: type\_t

**symbol**

*operator’s symbol. For example – operator+, symbol is equal to ‘+’*

**top\_parent**

Reference to top parent declaration.

@type: declaration\_t

**virtuality**

Describes the “virtuality” of the member (as defined by the string constants in the class :class:VIRTUALITY\_TYPES). @type: str

**class operator\_t** (\*args, \*\*keywords)

Bases: `pygccxml.declarations.calldef.calldef_t`

Base class for “operator” declarations.

Operators are constructs which behave like functions. Therefore, operator\_t has calldef\_t as parent class.

**OPERATOR\_WORD\_LEN = 8**

**argument\_types**

list of all argument types

**arguments**

The argument list. @type: list of argument\_t

**attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

**cache**

Implementation detail.

Reference to instance of algorithms\_cache\_t class.

**calling\_convention**

function calling convention. See :class:CALLING\_CONVENTION\_TYPES class for possible values

**create\_decl\_string** (with\_defaults=True)

**decl\_string**

Declaration full name.

**decorated\_name**

Unique declaration name extracted from a binary file ( .map, .dll, .so, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**demangled\_name**

returns function demangled name. It can help you to deal with function template instantiations

**does\_throw**

If False, than function does not throw any exception. In this case, function was declared with empty throw statement.

**exceptions**

The list of exceptions. @type: list of declaration\_t

**get\_mangled\_name** ()

**guess\_calling\_convention** ()

This function should be overridden in the derived classes and return more-or-less successfull guess about calling convention

**has\_ellipsis**

**has\_extern**

Was this callable declared as “extern”? @type: bool

**has\_inline**

Was this callable declared with “inline” specifier @type: bool

**i\_depend\_on\_them** (*recursive=True*)

**is\_artificial**  
Describes whether declaration is compiler generated or not  
@type: bool

**location**  
Location of the declaration within source file  
@type: location\_t

**mangled**  
Unique declaration name generated by the compiler.  
**Returns** the mangled name  
**Return type** str

**name**  
Declaration name @type: str

**optional\_args**  
list of all optional arguments, the arguments that have default value

**overloads**  
A list of overloaded “callables” (i.e. other callables with the same name within the same scope.  
@type: list of calldef\_t

**parent**  
Reference to parent declaration.  
@type: declaration\_t

**partial\_decl\_string**  
Declaration full name.

**partial\_name**  
Declaration name, without template default arguments.  
Right now std containers is the only classes that support this functionality.

**required\_args**  
list of all required arguments

**return\_type**  
The type of the return value of the “callable” or None (constructors). @type: type\_t

**symbol**  
*operator’s symbol. For example – operator+, symbol is equal to ‘+’*

**top\_parent**  
Reference to top parent declaration.  
@type: declaration\_t

### pygccxml.declarations.calldef\_types module

class **CALLING\_CONVENTION\_TYPES**

Bases: object

class that defines “calling convention” constants

**CDECL** = 'cdecl'

**FASTCALL** = 'fastcall'

**STDCALL** = 'stdcall'

**SYSTEM\_DEFAULT** = '<<<system default>>>'

**THISCALL** = 'thiscall'

**UNKNOWN** = ''

**all** = ('', 'cdecl', 'stdcall', 'thiscall', 'fastcall', '<<<system default>>>')

**static extract** (*text*, *default*='')

extracts calling convention from the text. If the calling convention could not be found, the "default" is used

**pattern** = <\_sre.SRE\_Pattern object at 0x1f24d10>

**FUNCTION\_VIRTUALITY\_TYPES**

alias of *VIRTUALITY\_TYPES*

**class VIRTUALITY\_TYPES**

Bases: object

class that defines "virtuality" constants

**ALL** = ['not virtual', 'virtual', 'pure virtual']

**NOT\_VIRTUAL** = 'not virtual'

**PURE\_VIRTUAL** = 'pure virtual'

**VIRTUAL** = 'virtual'

## pygccxml.declarations.class\_declaration module

defines classes, that describes C++ classes

**This modules contains definition for next C++ declarations:**

- class definition
- class declaration
- small helper class for describing C++ class hierarchy

**class ACCESS\_TYPES**

Bases: object

class that defines "access" constants

**ALL** = ['public', 'private', 'protected']

**PRIVATE** = 'private'

**PROTECTED** = 'protected'

**PUBLIC** = 'public'

**class CLASS\_TYPES**

Bases: object

class that defines "class" type constants

**ALL** = ['class', 'struct', 'union']



```
CLASS = 'class'
STRUCT = 'struct'
UNION = 'union'

class class_declaration_t (name='')
    Bases: pygccxml.declarations.declaration.declaration_t
    describes class declaration

    aliases
        List of aliases to this instance

    attributes
        GCCXML attributes, set using __attribute__((gccxml("...")))

        @type: str

    cache
        Implementation detail.

        Reference to instance of algorithms_cache_t class.

    create_decl_string (with_defaults=True)

    decl_string
        Declaration full name.

    decorated_name
        Unique declaration name extracted from a binary file ( .map, .dll, .so, etc ).

        @type: str

    demangled
        Declaration name, reconstructed from GCCXML generated unique name.

        @type: str

    get_mangled_name ()

    i_depend_on_them (recursive=True)

    is_artificial
        Describes whether declaration is compiler generated or not

        @type: bool

    location
        Location of the declaration within source file

        @type: location_t

    mangled
        Unique declaration name generated by the compiler.

        For GCCXML, you can get the mangled name for all the declarations. When using CastXML, calling
        mangled is only allowed on functions and variables. For other declarations it will raise an exception.

        Returns the mangled name

        Return type str

    name
        Declaration name @type: str
```

**parent**

Reference to parent declaration.

@type: `declaration_t`

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**top\_parent**

Reference to top parent declaration.

@type: `declaration_t`

**class** `class_t` (*name*='', *class\_type*='class', *is\_abstract*=False)

Bases: `pygccxml.declarations.scopeddef.scopeddef_t`, `pygccxml.declarations.byte_info.byte_info`, `pygccxml.declarations.elaborated_info.elaborated_info`

describes class definition

**ALLOW\_EMPTY\_MDECL\_WRAPPER** = False

**RECURSIVE\_DEFAULT** = True

**USE\_DEMANGLER\_AS\_NAME** = True

**adopt\_declaration** (*decl*, *access*)

adds new declaration to the class

**Parameters**

- **decl** – reference to a `declaration_t`
- **access** (:class:ACCESS\_TYPES) – member access type

**aliases**

List of aliases to this instance

**attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

**bases**

list of *base classes*

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**cache**

Implementation detail.

Reference to instance of `algorithms_cache_t` class.

**calldef** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)  
returns reference to “calldef” declaration, that is matched defined criteria

**calldefs** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)  
returns a set of `calldef_t` declarations, that are matched defined criteria

**casting\_operator** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)  
returns reference to casting operator declaration, that is matched defined criteria

**casting\_operators** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)  
returns a set of casting operator declarations, that are matched defined criteria

**class\_** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None*)  
returns reference to class declaration, that is matched defined criteria

**class\_type**  
describes class `type`

**classes** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)  
returns a set of class declarations, that are matched defined criteria

**clear\_optimizer** ()  
Cleans query optimizer state

**constructor** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)  
returns reference to constructor declaration, that is matched defined criteria

**constructors** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)  
returns a set of constructor declarations, that are matched defined criteria

**create\_decl\_string** (*with\_defaults=True*)

**decl** (*name=None, function=None, decl\_type=None, header\_dir=None, header\_file=None, recursive=None*)  
returns reference to declaration, that is matched defined criteria

**decl\_string**  
Declaration full name.

**declaration\_not\_found\_t**

**declarations**  
List of children declarations.

**Returns** List[`declarations.declaration_t`]

**decls** (*name=None, function=None, decl\_type=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)  
returns a set of declarations, that are matched defined criteria

**decorated\_name**  
Unique declaration name extracted from a binary file ( `.map`, `.dll`, `.so`, etc ).  
@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**derived**

list of *derived classes*

**elaborated\_type\_specifier**

*Elaborated specifier (can be – struct, union, class or enum).*

**Returns** elaborated specifier

**Return type** str

**enum** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None*)

Deprecated method. Use the enumeration() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**enumeration** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to enumeration declaration, that is matched defined criteria

**enumerations** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of enumeration declarations, that are matched defined criteria

**enums** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

Deprecated method. Use the enumerations() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**find\_out\_member\_access\_type** (*member*)

returns member access type

**Parameters** **member** (*declaration\_t*) – member of the class

**Return type** :class:ACCESS\_TYPES

**get\_mangled\_name** ()

**get\_members** (*access=None*)

returns list of members according to access type

If access equals to None, then returned list will contain all members. You should not modify the list content, otherwise different optimization data will stop work and may to give you wrong results.

**Parameters** **access** (:class:ACCESS\_TYPES) – describes desired members

**Return type** [ *members* ]

**i\_depend\_on\_them** (*recursive=True*)

**init\_optimizer** ()

Initializes query optimizer state.

**There are 4 internals hash tables:**

1. from type to declarations
2. from type to declarations for non-recursive queries
3. from type to name to declarations
4. from type to name to declarations for non-recursive queries

Almost every query includes declaration type information. Also very common query is to search some declaration(s) by name or full name. Those hash tables allows to search declaration very quick.

#### **is\_abstract**

describes whether class abstract or not

#### **is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

#### **location**

Location of the declaration within source file

@type: location\_t

#### **mangled**

Unique declaration name generated by the compiler.

For GCCXML, you can get the mangled name for all the declarations. When using CastXML, calling mangled is only allowed on functions and variables. For other declarations it will raise an exception.

**Returns** the mangled name

**Return type** str

**mem\_fun** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

Deprecated method. Use the member\_function() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**mem\_funcs** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

Deprecated method. Use the member\_functions() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**mem\_oper** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

Deprecated method. Use the member\_operator() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**mem\_opers** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

Deprecated method. Use the member\_operators() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**member\_function** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to member declaration, that is matched defined criteria

**member\_functions** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of member function declarations, that are matched defined criteria

**member\_operator** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to member operator declaration, that is matched defined criteria

**member\_operators** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of member operator declarations, that are matched defined criteria

**multiple\_declarations\_found\_t****name**

Declaration name @type: str

**operator** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to operator declaration, that is matched defined criteria

**operators** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of operator declarations, that are matched defined criteria

**parent**

Reference to parent declaration.

@type: declaration\_t

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**private\_members**

list of all private members

**protected\_members**

list of all protected members

**public\_members**

list of all public members

**recursive\_bases**

list of all *base classes*

**recursive\_derived**

list of all *derive classes*

**remove\_declaration** (*decl*)

removes decl from members list

**Parameters** **decl** (declaration\_t) – declaration to be removed

**top\_class**

reference to a parent class, which contains this class and defined within a namespace

if this class is defined under a namespace, self will be returned

**top\_parent**

Reference to top parent declaration.

@type: declaration\_t

**typedef** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to typedef declaration, that is matched defined criteria

**typedefs** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of typedef declarations, that are matched defined criteria

**use\_demangled\_as\_name**

**variable** (*name=None, function=None, decl\_type=None, header\_dir=None, header\_file=None, recursive=None*)  
 returns reference to variable declaration, that is matched defined criteria

**variables** (*name=None, function=None, decl\_type=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)  
 returns a set of variable declarations, that are matched defined criteria

**class dependency\_info\_t** (*decl, depend\_on\_it, access\_type=None, hint=None*)  
 Bases: object

**access\_type**

**decl**  
 Deprecated since 1.9.0. Will be removed in 2.0.0.

**declaration**

**depend\_on\_it**

**find\_out\_depend\_on\_it\_declarations** ()  
 If declaration depends on other declaration and not on some type this function will return reference to it.  
 Otherwise None will be returned

**hint**  
 The declaration, that report dependency can put some additional information about dependency. It can be used later

**static i\_depend\_on\_them** (*decl*)  
 Returns set of declarations. every item in the returned set, depends on a declaration from the input

**static we\_depend\_on\_them** (*decls*)  
 Returns set of declarations. every item in the returned set, depends on a declaration from the input

**get\_partial\_name** (*name*)

**class hierarchy\_info\_t** (*related\_class=None, access=None, is\_virtual=False*)  
 Bases: object

describes class relationship

**access**

**access\_type**  
 describes *hierarchy type*

**declaration\_path**

**declaration\_path\_hash**

**is\_virtual**  
 indicates whether the inheritance is virtual or not

**related\_class**  
 reference to base or derived *class*

**class impl\_details**  
 Bases: object

**static dig\_declarations** (*depend\_on\_it*)

## pygccxml.declarations.container\_traits module

Define a few algorithms that deal with different properties of std containers.

**all\_container\_traits** = (<pygccxml.declarations.container\_traits.container\_traits\_impl\_t object>, <pygccxml.declaration tuple of all STD container traits classes

**class container\_traits\_impl\_t** (*container\_name*, *element\_type\_index*, *element\_type\_typedef*,  
*eraser*, *key\_type\_index*=None, *key\_type\_typedef*=None, *unordered\_maps\_and\_sets*=False)

Bases: object

Implements the functionality needed for convenient work with STD container classes

**Implemented functionality:**

- find out whether a declaration is STD container or not
- find out container value( mapped ) type

This class tries to be useful as much as possible. For example, for class declaration (and not definition) it parses the class name in order to extract the information.

**class\_declaration** (*type\_*)  
Returns reference to the class declaration.

**element\_type** (*type\_*)  
returns reference to the class value mapped type declaration

**get\_container\_or\_none** (*type\_*)  
Returns reference to the class declaration or None.

**is\_mapping** (*type\_*)

**is\_my\_case** (*type\_*)  
Checks, whether type is STD container or not.

**is\_sequence** (*type\_*)

**key\_type** (*type\_*)  
returns reference to the class key type declaration

**name** ()

**remove\_defaults** (*type\_or\_string*)  
Removes template defaults from a templated class instantiation.

**For example:**

```
std::vector< int, std::allocator< int > >
```

**will become:**

```
std::vector< int >
```

**class defaults\_eraser** (*unordered\_maps\_and\_sets*)

Bases: object

**decorated\_call\_prefix** (*cls\_name*, *text*, *doit*)

**decorated\_call\_suffix** (*cls\_name*, *text*, *doit*)

**erase\_allocator** (*cls\_name*, *default\_allocator*='std::allocator')

**erase\_call** (*cls\_name*)

**erase\_compare\_allocator** (*cls\_name*, *default\_compare*='std::less', *default\_allocator*='std::allocator')

**erase\_container** (*cls\_name*, *default\_container\_name*='std::deque')



```

erase_container_compare (cls_name, default_container_name='std::vector', de-
                        fault_compare='std::less')
erase_hash_allocator (cls_name)
erase_hashmap_compare_allocator (cls_name)
erase_map_compare_allocator (cls_name, default_compare='std::less', de-
                        fault_allocator='std::allocator')
erase_recursive (cls_name)
no_const (cls_name)
no_end_const (cls_name)
no_gnustd (cls_name)
no_std (cls_name)
no_stdext (cls_name)
normalize (type_str)
replace_basic_string (cls_name)
find_container_traits (cls_or_string)
    Find the container traits type of a declaration.

    Parameters cls_or_string (str | declarations.declaration_t) – a string

    Returns a container traits

    Return type declarations.container_traits

sequential_container_traits = [<pygccxml.declarations.container_traits.container_traits_impl_t object>, <pygccxml.declarations.container_traits.container_traits_impl_t object>]
    list, that contains all STD container traits classes

```

## pygccxml.declarations.cpptypes module

defines classes, that describe C++ types

**FUNDAMENTAL\_TYPES** = {'short unsigned int': <pygccxml.declarations.cpptypes.short\_unsigned\_int\_t object>, 'jdouble': <pygccxml.declarations.cpptypes.jdouble\_t object>}  
 defines a mapping between fundamental type name and its synonym to the instance of class that describes the type

```

class array_t (base, size)
    Bases: pygccxml.declarations.cpptypes.compound_t
    represents C++ array type
    SIZE_UNKNOWN = -1
    base
        reference to internal/base class
    build_decl_string (with_defaults=True)
    byte_align
        Alignment of this declaration/type in bytes
        Returns Alignment of this declaration/type in bytes
        Return type int

```

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string****partial\_decl\_string****size**

returns array size

**class bool\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents bool type

**CPPNAME** = 'bool'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string****partial\_decl\_string****class calldef\_type\_t** (*return\_type=None, arguments\_types=None*)

Bases: object

base class for all types that describes “callable” declaration

**arguments\_types**

list of argument *types*

**has\_ellipsis****return\_type**

reference to *return type*

**class char\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents char type

**CPPNAME** = 'char'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string****partial\_decl\_string****class complex\_double\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents complex double type

**CPPNAME** = 'complex double'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string****partial\_decl\_string****class complex\_float\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents complex float type

**CPPNAME** = 'complex float'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class complex\_long\_double\_t**

Bases: `pygccxml.declarations.cpptypes.fundamental_t`

represents complex long double type

**CPPNAME** = 'complex long double'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class compound\_t** (*base*)

Bases: `pygccxml.declarations.cpptypes.type_t`

class that allows to represent compound types like `const int*`

**base**

reference to internal/base class

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

```

clone()
    returns new instance of the type

decl_string

partial_decl_string

class const_t (base)
    Bases: pygccxml.declarations.cpptypes.compound_t
    represents whatever const type

    base
        reference to internal/base class

    build_decl_string (with_defaults=True)

    byte_align
        Alignment of this declaration/type in bytes

        Returns Alignment of this declaration/type in bytes

        Return type int

    byte_size
        Size of this declaration/type in bytes

        Returns Size of this declaration/type in bytes

        Return type int

    clone()
        returns new instance of the type

    decl_string

    partial_decl_string

class declared_t (declaration)
    Bases: pygccxml.declarations.cpptypes.type_t, pygccxml.declarations.  

byte_info.byte_info
    class that binds between to hierarchies: type_t and declaration_t

    build_decl_string (with_defaults=True)

    byte_align
        Alignment of this declaration/type in bytes

        Returns Alignment of this declaration/type in bytes

        Return type int

    byte_size
        Size of this declaration/type in bytes

        Returns Size of this declaration/type in bytes

        Return type int

    clone()
        returns new instance of the type

    decl_string

    declaration
        reference to declaration_t

```

**partial\_decl\_string**

**class double\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents double type

**CPPNAME** = 'double'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone** ()

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class dummy\_type\_t** (*decl\_string*)

Bases: *pygccxml.declarations.cpptypes.type\_t*

provides *type\_t* interface for a string, that defines C++ type.

This class could be very useful in the code generator.

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone** ()

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class elaborated\_t** (*base*)

Bases: *pygccxml.declarations.cpptypes.compound\_t*

represents *elaborated* type

**base**  
reference to internal/base class

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**  
Alignment of this declaration/type in bytes  
**Returns** Alignment of this declaration/type in bytes  
**Return type** int

**byte\_size**  
Size of this declaration/type in bytes  
**Returns** Size of this declaration/type in bytes  
**Return type** int

**clone** ()  
returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class ellipsis\_t**  
Bases: *pygccxml.declarations.cpptypes.type\_t*  
type, that represents "..." in function definition  
**build\_decl\_string** (*with\_defaults=True*)  
**byte\_align**  
Alignment of this declaration/type in bytes  
**Returns** Alignment of this declaration/type in bytes  
**Return type** int  
**byte\_size**  
Size of this declaration/type in bytes  
**Returns** Size of this declaration/type in bytes  
**Return type** int  
**clone** ()  
returns new instance of the type  
**decl\_string**  
**partial\_decl\_string**

**class float\_t**  
Bases: *pygccxml.declarations.cpptypes.fundamental\_t*  
represents float type  
**CPPNAME** = 'float'  
**build\_decl\_string** (*with\_defaults=True*)  
**byte\_align**  
Alignment of this declaration/type in bytes  
**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class free\_function\_type\_t** (*return\_type=None, arguments\_types=None*)

Bases: *pygccxml.declarations.cpptypes.type\_t, pygccxml.declarations.cpptypes.calldef\_type\_t*

describes free function type

**NAME\_TEMPLATE** = '%(return\_type)s (\*) ( %(arguments)s )'

**TYPEDEF\_NAME\_TEMPLATE** = '%(return\_type)s ( \*%(typedef\_name)s ) ( %(arguments)s )'

**arguments\_types**

list of argument *types*

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**static create\_decl\_string** (*return\_type, arguments\_types, with\_defaults=True*)

Returns free function type

**Parameters**

- **return\_type** (*type\_t*) – function return type
- **arguments\_types** – list of argument *type*

**Return type** *free\_function\_type\_t*

**create\_typedef** (*typedef\_name, unused=None, with\_defaults=True*)

returns string, that contains valid C++ code, that defines typedef to function type

**Parameters** **name** – the desired name of typedef

**decl\_string**

**has\_ellipsis**



**partial\_decl\_string**

**return\_type**

reference to *return type*

**class fundamental\_t** (*name*)

Bases: *pygccxml.declarations.cpptypes.type\_t*

base class for all fundamental, build-in types

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone** ()

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class int128\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents \_\_int128\_t type

**CPPNAME** = '\_\_int128\_t'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone** ()

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class int\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents int type

```
CPPNAME = 'int'
build_decl_string (with_defaults=True)
byte_align
    Alignment of this declaration/type in bytes
    Returns Alignment of this declaration/type in bytes
    Return type int
byte_size
    Size of this declaration/type in bytes
    Returns Size of this declaration/type in bytes
    Return type int
clone ()
    returns new instance of the type
decl_string
partial_decl_string
class java_fundamental_t (name)
    Bases: pygccxml.declarations.cpptypes.fundamental_t
    base class for all JNI defined fundamental types
    build_decl_string (with_defaults=True)
    byte_align
        Alignment of this declaration/type in bytes
        Returns Alignment of this declaration/type in bytes
        Return type int
    byte_size
        Size of this declaration/type in bytes
        Returns Size of this declaration/type in bytes
        Return type int
    clone ()
        returns new instance of the type
    decl_string
    partial_decl_string
class jboolean_t
    Bases: pygccxml.declarations.cpptypes.java_fundamental_t
    represents jboolean type
    JNAME = 'jboolean'
    build_decl_string (with_defaults=True)
    byte_align
        Alignment of this declaration/type in bytes
        Returns Alignment of this declaration/type in bytes
        Return type int
```

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class jbyte\_t**

Bases: *pygccxml.declarations.cpptypes.java\_fundamental\_t*

represents jbyte type

**JNAME** = 'jbyte'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class jchar\_t**

Bases: *pygccxml.declarations.cpptypes.java\_fundamental\_t*

represents jchar type

**JNAME** = 'jchar'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

```
clone ()
    returns new instance of the type

decl_string
partial_decl_string

class jdouble_t
    Bases: pygccxml.declarations.cpptypes.java_fundamental_t
    represents jdouble type
    JNAME = 'jdouble'
    build_decl_string (with_defaults=True)
    byte_align
        Alignment of this declaration/type in bytes
        Returns Alignment of this declaration/type in bytes
        Return type int
    byte_size
        Size of this declaration/type in bytes
        Returns Size of this declaration/type in bytes
        Return type int
    clone ()
        returns new instance of the type
    decl_string
    partial_decl_string

class jfloat_t
    Bases: pygccxml.declarations.cpptypes.java_fundamental_t
    represents jfloat type
    JNAME = 'jfloat'
    build_decl_string (with_defaults=True)
    byte_align
        Alignment of this declaration/type in bytes
        Returns Alignment of this declaration/type in bytes
        Return type int
    byte_size
        Size of this declaration/type in bytes
        Returns Size of this declaration/type in bytes
        Return type int
    clone ()
        returns new instance of the type
    decl_string
    partial_decl_string
```

```
class jint_t
    Bases: pygccxml.declarations.cpptypes.java_fundamental_t
    represents jint type
    JNAME = 'jint'
    build_decl_string (with_defaults=True)
    byte_align
        Alignment of this declaration/type in bytes
        Returns Alignment of this declaration/type in bytes
        Return type int
    byte_size
        Size of this declaration/type in bytes
        Returns Size of this declaration/type in bytes
        Return type int
    clone ()
        returns new instance of the type
    decl_string
    partial_decl_string

class jlong_t
    Bases: pygccxml.declarations.cpptypes.java_fundamental_t
    represents jlong type
    JNAME = 'jlong'
    build_decl_string (with_defaults=True)
    byte_align
        Alignment of this declaration/type in bytes
        Returns Alignment of this declaration/type in bytes
        Return type int
    byte_size
        Size of this declaration/type in bytes
        Returns Size of this declaration/type in bytes
        Return type int
    clone ()
        returns new instance of the type
    decl_string
    partial_decl_string

class jshort_t
    Bases: pygccxml.declarations.cpptypes.java_fundamental_t
    represents jshort type
    JNAME = 'jshort'
    build_decl_string (with_defaults=True)
```

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string****partial\_decl\_string****class long\_double\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents long double type

**CPPNAME** = 'long double'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string****partial\_decl\_string****class long\_int\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents long int type

**CPPNAME** = 'long int'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**  
Size of this declaration/type in bytes  
**Returns** Size of this declaration/type in bytes  
**Return type** int

**clone()**  
returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class long\_long\_int\_t**  
Bases: *pygccxml.declarations.cpptypes.fundamental\_t*  
represents long long int type  
**CPPNAME** = 'long long int'  
**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**  
Alignment of this declaration/type in bytes  
**Returns** Alignment of this declaration/type in bytes  
**Return type** int

**byte\_size**  
Size of this declaration/type in bytes  
**Returns** Size of this declaration/type in bytes  
**Return type** int

**clone()**  
returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class long\_long\_unsigned\_int\_t**  
Bases: *pygccxml.declarations.cpptypes.fundamental\_t*  
represents long long unsigned int type  
**CPPNAME** = 'long long unsigned int'  
**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**  
Alignment of this declaration/type in bytes  
**Returns** Alignment of this declaration/type in bytes  
**Return type** int

**byte\_size**  
Size of this declaration/type in bytes  
**Returns** Size of this declaration/type in bytes  
**Return type** int

**clone()**  
returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class long\_unsigned\_int\_t**  
Bases: *pygccxml.declarations.cpptypes.fundamental\_t*  
represents long unsigned int type  
**CPPNAME** = 'long unsigned int'  
**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**  
Alignment of this declaration/type in bytes  
**Returns** Alignment of this declaration/type in bytes  
**Return type** int

**byte\_size**  
Size of this declaration/type in bytes  
**Returns** Size of this declaration/type in bytes  
**Return type** int

**clone()**  
returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class member\_function\_type\_t** (*class\_inst=None, return\_type=None, arguments\_types=None, has\_const=False*)  
Bases: *pygccxml.declarations.cpptypes.type\_t, pygccxml.declarations.cpptypes.calldef\_type\_t*  
describes member function type  
**NAME\_TEMPLATE** = '%(return\_type)s ( %(class)s::\* )( %(arguments)s )%(has\_const)s'  
**TYPEDDEF\_NAME\_TEMPLATE** = '%(return\_type)s ( %(class)s::\*%(typedef\_name)s)( %(arguments)s ) %(has\_const)s'  
**arguments\_types**  
list of argument *types*  
**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**  
Alignment of this declaration/type in bytes  
**Returns** Alignment of this declaration/type in bytes  
**Return type** int

**byte\_size**  
Size of this declaration/type in bytes  
**Returns** Size of this declaration/type in bytes  
**Return type** int



**class\_inst**  
reference to parent `class`

**clone()**  
returns new instance of the type

**static create\_decl\_string** (*return\_type*, *class\_decl\_string*, *arguments\_types*, *has\_const*,  
*with\_defaults=True*)

**create\_typedef** (*typedef\_name*, *class\_alias=None*, *with\_defaults=True*)  
creates typedef to the function type

**Parameters** *typedef\_name* – desired type name

**Return type** `string`

**decl\_string**

**has\_const**  
describes, whether function has `const` modifier

**has\_ellipsis**

**partial\_decl\_string**

**return\_type**  
reference to *return type*

**class member\_variable\_type\_t** (*class\_inst=None*, *variable\_type=None*)  
Bases: `pygccxml.declarations.cpptypes.compound_t`  
describes member variable type

**NAME\_TEMPLATE** = '%(type)s ( %(class)s::\* )'

**base**  
reference to internal/base class

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**  
Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** `int`

**byte\_size**  
Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** `int`

**clone()**  
returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**variable\_type**  
describes member variable *type*

**class pointer\_t** (*base*)  
Bases: `pygccxml.declarations.cpptypes.compound_t`  
represents *whatever\** type

**base**  
reference to internal/base class

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**  
Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**  
Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone** ()  
returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class reference\_t** (*base*)  
Bases: [\*pygccxml.declarations.cpptypes.compound\\_t\*](#)  
represents *whatever*& type

**base**  
reference to internal/base class

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**  
Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**  
Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone** ()  
returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class restrict\_t** (*base*)  
Bases: [\*pygccxml.declarations.cpptypes.compound\\_t\*](#)  
represents *restrict whatever* type

**base**  
reference to internal/base class

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string****partial\_decl\_string****class short\_int\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents short int type

**CPPNAME** = 'short int'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string****partial\_decl\_string****class short\_unsigned\_int\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents short unsigned int type

**CPPNAME** = 'short unsigned int'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class signed\_char\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents signed char type

**CPPNAME** = 'signed char'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class type\_qualifiers\_t** (*has\_static=False, has\_mutable=False, has\_extern=False*)

Bases: object

contains additional information about type: mutable, static, extern

**has\_extern**

**has\_mutable**

**has\_static**

**class type\_t**

Bases: *pygccxml.declarations.byte\_info.byte\_info*

base class for all types

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string****partial\_decl\_string****class uint128\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents \_\_uint128\_t type

**CPPNAME** = '\_\_uint128\_t'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string****partial\_decl\_string****class unknown\_t**

Bases: *pygccxml.declarations.cpptypes.type\_t*

type, that represents all C++ types, that could not be parsed by GCC-XML

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class unsigned\_char\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents unsigned char type

**CPPNAME = 'unsigned char'**

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone ()**

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class unsigned\_int\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents unsigned int type

**CPPNAME = 'unsigned int'**

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone ()**

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class void\_t**

Bases: *pygccxml.declarations.cpptypes.fundamental\_t*

represents void type

**CPPNAME** = 'void'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**  
Alignment of this declaration/type in bytes  
**Returns** Alignment of this declaration/type in bytes  
**Return type** int

**byte\_size**  
Size of this declaration/type in bytes  
**Returns** Size of this declaration/type in bytes  
**Return type** int

**clone** ()  
returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class volatile\_t** (*base*)  
Bases: *pygccxml.declarations.cpptypes.compound\_t*  
represents *volatile whatever* type

**base**  
reference to internal/base class

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**  
Alignment of this declaration/type in bytes  
**Returns** Alignment of this declaration/type in bytes  
**Return type** int

**byte\_size**  
Size of this declaration/type in bytes  
**Returns** Size of this declaration/type in bytes  
**Return type** int

**clone** ()  
returns new instance of the type

**decl\_string**

**partial\_decl\_string**

**class wchar\_t**  
Bases: *pygccxml.declarations.cpptypes.fundamental\_t*  
represents *wchar\_t* type

**CPPNAME** = 'wchar\_t'

**build\_decl\_string** (*with\_defaults=True*)

**byte\_align**  
Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

**byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**clone()**

returns new instance of the type

**decl\_string**

**partial\_decl\_string**

## pygccxml.declarations.decl\_factory module

defines default declarations factory class

**class decl\_factory\_t**

Bases: object

declarations factory class

**create\_casting\_operator** (\*arguments, \*\*keywords)

creates instance of class that describes casting operator declaration

**create\_class** (\*arguments, \*\*keywords)

creates instance of class that describes class definition declaration

**create\_class\_declaration** (\*arguments, \*\*keywords)

creates instance of class that describes class declaration

**create\_constructor** (\*arguments, \*\*keywords)

creates instance of class that describes constructor declaration

**create\_destructor** (\*arguments, \*\*keywords)

creates instance of class that describes destructor declaration

**create\_enumeration** (\*arguments, \*\*keywords)

creates instance of class that describes enumeration declaration

**create\_free\_function** (\*arguments, \*\*keywords)

creates instance of class that describes free function declaration

**create\_free\_operator** (\*arguments, \*\*keywords)

creates instance of class that describes free operator declaration

**create\_member\_function** (\*arguments, \*\*keywords)

creates instance of class that describes member function declaration

**create\_member\_operator** (\*arguments, \*\*keywords)

creates instance of class that describes member operator declaration

**create\_namespace** (\*arguments, \*\*keywords)

creates instance of class that describes namespace declaration

**create\_typedef** (\*arguments, \*\*keywords)

creates instance of class that describes typedef declaration



**create\_variable** (*\*arguments*, *\*\*keywords*)  
creates instance of class that describes variable declaration

### pygccxml.declarations.decl\_printer module

defines class, *decl\_printer\_t* that prints declarations tree in a user friendly format

**class decl\_printer\_t** (*level=0*, *print\_details=True*, *recursive=True*, *writer=<function \_stdout\_writer>*,  
*verbose=True*)

Bases: *pygccxml.declarations.decl\_visitor.decl\_visitor\_t*

helper class for printing declarations tree

**INDENT\_SIZE = 4**

**JUSTIFY = 20**

**clone** (*increment\_level=True*)

**instance**

**static is\_builtin\_decl** (*decl*)

**level**

**print\_calldef\_info** (*decl=None*)

**print\_decl\_header** ()

**print\_details**

**recursive**

**verbose**

**visit\_casting\_operator** ()

**visit\_class** ()

**visit\_class\_declaration** ()

**visit\_constructor** ()

**visit\_destructor** ()

**visit\_enumeration** ()

**visit\_free\_function** ()

**visit\_free\_operator** ()

**visit\_member\_function** ()

**visit\_member\_operator** ()

**visit\_namespace** ()

**visit\_typedef** ()

**visit\_variable** ()

**writer**

**dump\_declarations** (*declarations*, *file\_path*)

Dump declarations tree rooted at each of the included nodes to the file

**Parameters**

- **declarations** – either a single :class:declaration\_t object or a list of :class:declaration\_t objects
- **file\_path** – path to a file

**print\_declarations**(*decls*, *detailed=True*, *recursive=True*, *writer=<function <lambda>>*, *verbose=True*)

print declarations tree rooted at each of the included nodes.

**Parameters** **decls** – either a single :class:declaration\_t object or list of :class:declaration\_t objects

## pygccxml.declarations.decl\_visitor module

defines declarations visitor class interface

**class decl\_visitor\_t**

Bases: object

declarations visitor interface

All functions within this class should be redefined in derived classes.

**visit\_casting\_operator()**

**visit\_class()**

**visit\_class\_declaration()**

**visit\_constructor()**

**visit\_destructor()**

**visit\_enumeration()**

**visit\_free\_function()**

**visit\_free\_operator()**

**visit\_member\_function()**

**visit\_member\_operator()**

**visit\_namespace()**

**visit\_typedef()**

**visit\_variable()**

## pygccxml.declarations.declaration module

Defines pygccxml.declarations.declaration\_t class - all declarations base class.

**class declaration\_t**(*name='', location=None, is\_artificial=False, mangled=None, demangled=None, attributes=None*)

Bases: object

Base class for all classes that represent a C++ declaration.

**attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

**cache**

Implementation detail.

Reference to instance of `algorithms_cache_t` class.

**create\_decl\_string** (*with\_defaults=True*)**decl\_string**

Declaration full name.

**decorated\_name**

Unique declaration name extracted from a binary file ( `.map`, `.dll`, `.so`, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**get\_mangled\_name** ()**i\_depend\_on\_them** (*recursive=True*)

Return list of all types and declarations the declaration depends on

**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: `location_t`

**mangled**

Unique declaration name generated by the compiler.

For GCCXML, you can get the mangled name for all the declarations. When using CastXML, calling `mangled` is only allowed on functions and variables. For other declarations it will raise an exception.

**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**parent**

Reference to parent declaration.

@type: `declaration_t`

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now `std containers` is the only classes that support this functionality.

**top\_parent**

Reference to top parent declaration.

@type: `declaration_t`

## pygccxml.declarations.declaration\_utils module

**declaration\_path** (*decl*, *with\_defaults=None*)

Returns a list of parent declarations names.

**Parameters** **decl** (*declaration\_t*) – declaration for which declaration path should be calculated.

**Returns**

**list of names, where first item is the top** parent name and last item the inputted declaration name.

**Return type** list[(str | basestring)]

**full\_name** (*decl*, *with\_defaults=True*)

Returns declaration full qualified name.

If *decl* belongs to anonymous namespace or class, the function will return C++ illegal qualified name.

**Parameters** **decl** (*declaration\_t*) – declaration for which the full qualified name should be calculated.

**Returns** full name of the declaration.

**Return type** list[(str | basestring)]

**full\_name\_from\_declaration\_path** (*dpath*)

**get\_named\_parent** (*decl*)

Returns a reference to a named parent declaration.

**Parameters** **decl** (*declaration\_t*) – the child declaration

**Returns** the declaration or None if not found.

**Return type** *declaration\_t*

**partial\_declaration\_path** (*decl*)

Returns a list of parent declarations names without template arguments that have default value.

**Parameters** **decl** (*declaration\_t*) – declaration for which the partial declaration path should be calculated.

**Returns**

**list of names, where first item is the top** parent name and last item the inputted declaration name.

**Return type** list[(str | basestring)]

## pygccxml.declarations.declarations\_matchers module

**class** **calldef\_matcher\_t** (*name=None*, *return\_type=None*, *arg\_types=None*, *decl\_type=None*,  
*header\_dir=None*, *header\_file=None*)

Bases: *pygccxml.declarations.declarations\_matchers.declaration\_matcher\_t*

**Instance of this class will match callable by the following criteria:**

- *declaration\_matcher\_t* criteria
- return type. For example: *int\_t* or 'int'
- argument types

```

    check_name (decl)
    decl_name_only
    is_full_name ()
    name

```

class **declaration\_matcher\_t** (*name=None, decl\_type=None, header\_dir=None, header\_file=None*)  
 Bases: *pygccxml.declarations.matchers.matcher\_base\_t*

Instance of this class will match declarations by next criteria:

- declaration name, also could be fully qualified name Example: *wstring* or *::std::wstring*
- declaration type Example: *class\_t*, *namespace\_t*, *enumeration\_t*
- location within file system ( file or directory )

```

    check_name (decl)
    decl_name_only
    is_full_name ()
    name

```

class **namespace\_matcher\_t** (*name=None*)  
 Bases: *pygccxml.declarations.declarations\_matchers.declaration\_matcher\_t*

Instance of this class will match namespaces by name.

```

    check_name (decl)
    decl_name_only
    is_full_name ()
    name

```

class **operator\_matcher\_t** (*name=None, symbol=None, return\_type=None, arg\_types=None, decl\_type=None, header\_dir=None, header\_file=None*)  
 Bases: *pygccxml.declarations.declarations\_matchers.calldef\_matcher\_t*

Instance of this class will match operators by next criteria:

- *calldef\_matcher\_t* criteria
- operator symbol: *=*, *!=*, *()*, *[]* and etc

```

    check_name (decl)
    decl_name_only
    is_full_name ()
    name

```

class **variable\_matcher\_t** (*name=None, decl\_type=None, header\_dir=None, header\_file=None*)  
 Bases: *pygccxml.declarations.declarations\_matchers.declaration\_matcher\_t*

Instance of this class will match variables by next criteria:

- *declaration\_matcher\_t* criteria
- variable type. Example: *int\_t* or *'int'*

```

    check_name (decl)
    decl_name_only

```

```
is_full_name()
```

```
name
```

### pygccxml.declarations.elaborated\_info module

```
class elaborated_info (elaborated_type_specifier)
```

Bases: object

This class stores the name of the elaborated type specifier.

```
elaborated_type_specifier
```

*Elaborated specifier (can be – struct, union, class or enum).*

**Returns** elaborated specifier

**Return type** str

### pygccxml.declarations.enumeration module

defines class, that describes C++ *enum*

```
class enumeration_t (name='', values=None)
```

Bases: `pygccxml.declarations.declaration.declaration_t`, `pygccxml.declarations.byte_info.byte_info`, `pygccxml.declarations.elaborated_info.elaborated_info`

describes C++ *enum*

```
append_value (valuenam, valuenum=None)
```

Append another enumeration value to the *enum*.

The numeric value may be None in which case it is automatically determined by increasing the value of the last item.

When the 'values' attribute is accessed the resulting list will be in the same order as `append_value()` was called.

#### Parameters

- **valuenam** (*str*) – The name of the value.
- **valuenum** (*int*) – The numeric value or None.

#### attributes

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

#### byte\_align

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

#### byte\_size

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

**cache**

Implementation detail.

Reference to instance of `algorithms_cache_t` class.

**create\_decl\_string** (*with\_defaults=True*)**decl\_string**

Declaration full name.

**decorated\_name**

Unique declaration name extracted from a binary file ( `.map`, `.dll`, `.so`, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**elaborated\_type\_specifier**

*Elaborated specifier (can be – struct, union, class or enum).*

**Returns** elaborated specifier

**Return type** str

**get\_mangled\_name** ()**get\_name2value\_dict** ()

returns a dictionary, that maps between *enum* name( key ) and *enum* value( value )

**has\_value\_name** (*name*)

Check if this *enum* has a particular name among its values.

**Parameters** **name** (*str*) – Enumeration value name

**Return type** True if there is an enumeration value with the given name

**i\_depend\_on\_them** (*recursive=True*)**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: `location_t`

**mangled**

Unique declaration name generated by the compiler.

For GCCXML, you can get the mangled name for all the declarations. When using CastXML, calling `mangled` is only allowed on functions and variables. For other declarations it will raise an exception.

**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**parent**

Reference to parent declaration.

**@type:** declaration\_t

**partial\_decl\_string**  
Declaration full name.

**partial\_name**  
Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**top\_parent**  
Reference to top parent declaration.

**@type:** declaration\_t

**values**  
A list of tuples (valname(str), valnum(int)) that contain the enumeration values. @type: list

### pygccxml.declarations.free\_calldef module

**class free\_calldef\_t** (\*args, \*\*keywords)  
Bases: `pygccxml.declarations.calldef.calldef_t`  
base class for “callable” declarations that defined within C++ namespace

**argument\_types**  
list of all argument types

**arguments**  
The argument list. @type: list of argument\_t

**attributes**  
GCCXML attributes, set using `__attribute__((gccxml("...")))`  
@type: str

**cache**  
Implementation detail.  
Reference to instance of algorithms\_cache\_t class.

**calling\_convention**  
function calling convention. See :class:CALLING\_CONVENTION\_TYPES class for possible values

**create\_decl\_string** (with\_defaults=True)

**decl\_string**  
Declaration full name.

**decorated\_name**  
Unique declaration name extracted from a binary file ( .map, .dll, .so, etc ).  
@type: str

**demangled**  
Declaration name, reconstructed from GCCXML generated unique name.  
@type: str

**demangled\_name**  
returns function demangled name. It can help you to deal with function template instantiations



**does\_throw**

If False, than function does not throw any exception. In this case, function was declared with empty throw statement.

**exceptions**

The list of exceptions. @type: list of `declaration_t`

**function\_type()**

returns function type. See `type_t` hierarchy

**get\_mangled\_name()****guess\_calling\_convention()**

This function should be overridden in the derived classes and return more-or-less successfull guess about calling convention

**has\_ellipsis****has\_extern**

Was this callable declared as “extern”? @type: bool

**has\_inline**

Was this callable declared with “inline” specifier @type: bool

**i\_depend\_on\_them** (*recursive=True*)**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: `location_t`

**mangled**

Unique declaration name generated by the compiler.

**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**optional\_args**

list of all optional arguments, the arguments that have default value

**overloads**

A list of overloaded “callables” (i.e. other callables with the same name within the same scope.

@type: list of `calldef_t`

**parent**

Reference to parent declaration.

@type: `declaration_t`

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**required\_args**

list of all required arguments

**return\_type**

The type of the return value of the “callable” or None (constructors). @type: `type_t`

**top\_parent**

Reference to top parent declaration.

@type: `declaration_t`

**class `free_function_t` (\*args, \*\*keywords)**

Bases: `pygccxml.declarations.free_calldef.free_calldef_t`

describes free function declaration

**argument\_types**

list of all argument types

**arguments**

The argument list. @type: list of `argument_t`

**attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

**cache**

Implementation detail.

Reference to instance of `algorithms_cache_t` class.

**calling\_convention**

function calling convention. See `:class:CALLING_CONVENTION_TYPES` class for possible values

**`create_decl_string` (*with\_defaults=True*)**

**decl\_string**

Declaration full name.

**decorated\_name**

Unique declaration name extracted from a binary file ( `.map`, `.dll`, `.so`, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**demangled\_name**

returns function demangled name. It can help you to deal with function template instantiations

**does\_throw**

If False, than function does not throw any exception. In this case, function was declared with empty throw statement.

**exceptions**

The list of exceptions. @type: list of `declaration_t`

**function\_type ()**

returns function type. See `type_t` hierarchy

**get\_mangled\_name ()**

**guess\_calling\_convention()**

This function should be overridden in the derived classes and return more-or-less successful guess about calling convention

**has\_ellipsis**

**has\_extern**

Was this callable declared as “extern”? @type: bool

**has\_inline**

Was this callable declared with “inline” specifier @type: bool

**i\_depend\_on\_them** (*recursive=True*)

**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: location\_t

**mangled**

Unique declaration name generated by the compiler.

**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**optional\_args**

list of all optional arguments, the arguments that have default value

**overloads**

A list of overloaded “callables” (i.e. other callables with the same name within the same scope.

@type: list of calldef\_t

**parent**

Reference to parent declaration.

@type: declaration\_t

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**required\_args**

list of all required arguments

**return\_type**

The type of the return value of the “callable” or None (constructors). @type: type\_t

**top\_parent**

Reference to top parent declaration.

@type: declaration\_t

**class free\_operator\_t** (\*args, \*\*keywords)  
Bases: `pygccxml.declarations.free_calldef.free_calldef_t`, `pygccxml.declarations.calldef_members.operator_t`  
describes free operator declaration  
**OPERATOR\_WORD\_LEN** = 8  
**argument\_types**  
list of all argument types  
**arguments**  
The argument list. @type: list of `argument_t`  
**attributes**  
GCCXML attributes, set using `__attribute__((gccxml("...")))`  
@type: str  
**cache**  
Implementation detail.  
Reference to instance of `algorithms_cache_t` class.  
**calling\_convention**  
function calling convention. See :class:CALLING\_CONVENTION\_TYPES class for possible values  
**class\_types**  
list of class/class declaration types, extracted from the operator arguments  
**create\_decl\_string** (with\_defaults=True)  
**decl\_string**  
Declaration full name.  
**decorated\_name**  
Unique declaration name extracted from a binary file ( .map, .dll, .so, etc ).  
@type: str  
**demangled**  
Declaration name, reconstructed from GCCXML generated unique name.  
@type: str  
**demangled\_name**  
returns function demangled name. It can help you to deal with function template instantiations  
**does\_throw**  
If False, than function does not throw any exception. In this case, function was declared with empty throw statement.  
**exceptions**  
The list of exceptions. @type: list of `declaration_t`  
**function\_type** ()  
returns function type. See `type_t` hierarchy  
**get\_mangled\_name** ()  
**guess\_calling\_convention** ()  
This function should be overridden in the derived classes and return more-or-less successfull guess about calling convention  
**has\_ellipsis**

**has\_extern**

Was this callable declared as “extern”? @type: bool

**has\_inline**

Was this callable declared with “inline” specifier @type: bool

**i\_depend\_on\_them** (*recursive=True*)**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: location\_t

**mangled**

Unique declaration name generated by the compiler.

**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**optional\_args**

list of all optional arguments, the arguments that have default value

**overloads**

A list of overloaded “callables” (i.e. other callables with the same name within the same scope.

@type: list of calldef\_t

**parent**

Reference to parent declaration.

@type: declaration\_t

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**required\_args**

list of all required arguments

**return\_type**

The type of the return value of the “callable” or None (constructors). @type: type\_t

**symbol**

*operator’s symbol. For example – operator+, symbol is equal to ‘+’*

**top\_parent**

Reference to top parent declaration.

@type: declaration\_t

### pygccxml.declarations.function\_traits module

defines few algorithms, that deals with different properties of functions

**is\_same\_function** (*f1, f2*)  
returns true if f1 and f2 is same function

Use case: sometimes when user defines some virtual function in base class, it overrides it in a derived one. Sometimes we need to know whether two member functions is actually same function.

**is\_same\_return\_type** (*f1, f2*)

### pygccxml.declarations.has\_operator\_matcher module

**has\_public\_binary\_operator** (*type\_, operator\_symbol*)  
returns True, if *type\_* has public binary operator, otherwise False

**has\_public\_equal** (*decl\_type*)  
returns True, if class has public operator==, otherwise False

**has\_public\_less** (*decl\_type*)  
returns True, if class has public operator<, otherwise False

### pygccxml.declarations.location module

**class location\_t** (*file\_name='', line=-1*)  
Bases: object

Provides information about the location of the declaration within the source file.

**as\_tuple** ()  
Return tuple(self.file\_name, self.line)

**file\_name**  
Absolute source file name, type string.

**line**  
Line number, type int.

### pygccxml.declarations.matchers module

defines all “built-in” classes that implement declarations compare functionality according to some criteria

**class access\_type\_matcher\_t** (*access\_type*)  
Bases: *pygccxml.declarations.matchers.matcher\_base\_t*

Instance of this class will match declaration by its access type: public, private or protected. If declarations does not have access type, for example free function, then *False* will be returned.

**class and\_matcher\_t** (*matchers*)  
Bases: *pygccxml.declarations.matchers.matcher\_base\_t*

Combine several other matchers with “&” (and) operator.

For example: find all private functions with name XXX

```
matcher = access_type_matcher_t( 'private' ) & calldef_matcher_t( name=  
↳ 'XXX' )
```

**class custom\_matcher\_t** (*function*)

Bases: `pygccxml.declarations.matchers.matcher_base_t`

Instance of this class will match declaration by user custom criteria.

**class matcher\_base\_t**

Bases: `object`

`matcher_base_t` class defines interface for classes that will implement compare functionality according to some criteria.

**class not\_matcher\_t** (*matcher*)

Bases: `pygccxml.declarations.matchers.matcher_base_t`

return the inverse result of a matcher

For example: find all public and protected declarations

```
matcher = ~access_type_matcher_t( 'private' )
```

**class or\_matcher\_t** (*matchers*)

Bases: `pygccxml.declarations.matchers.matcher_base_t`

Combine several other matchers with “|” (or) operator.

For example: find all functions and variables with name ‘XXX’

```
matcher = variable_matcher_t( name='XXX' ) | calldef_matcher_t( name=
↳ 'XXX' )
```

**class regex\_matcher\_t** (*regex, function=None*)

Bases: `pygccxml.declarations.matchers.matcher_base_t`

Instance of this class will match declaration using regular expression. User should supply a function that will extract from declaration desired information as string. Later, this matcher will match that string using user regular expression.

**class virtuality\_type\_matcher\_t** (*virtuality\_type*)

Bases: `pygccxml.declarations.matchers.matcher_base_t`

Instance of this class will match declaration by its virtual type: not virtual, virtual or pure virtual. If declarations does not have “virtual” property, for example free function, then *False* will be returned.

## pygccxml.declarations.mdecl\_wrapper module

defines class `mdecl_wrapper_t` that allows to work on set of declarations, as it was one declaration.

The `class` allows user to not write “for” loops within the code.

**class call\_redirector\_t** (*name, decls*)

Bases: `object`

Internal class used to call some function of objects

**class mdecl\_wrapper\_t** (*decls*)

Bases: `object`

multiple declarations class wrapper

The main purpose of this class is to allow an user to work on many declarations, as they were only one single declaration.

For example, instead of writing *for* loop like the following

```
for c in global_namespace.classes():
    c.attribute = "xxxx"
```

you can write:

```
global_namespace.classes().attribute = "xxxx"
```

The same functionality could be applied on “set” methods too.

`to_list()`

## pygccxml.declarations.namespace module

Describe a C++ namespace declaration.

**get\_global\_namespace** (*decls*)

Get the global namespace (::) from a declaration tree.

**Parameters** *decls* (*list[declaration\_t]*) – a list of declarations

**Returns** the global namespace\_t object (::)

**Return type** *namespace\_t*

**class namespace\_t** (*name='', declarations=None*)

Bases: *pygccxml.declarations.scopeddef.scopeddef\_t*

Describes C++ namespace.

**ALLOW\_EMPTY\_MDECL\_WRAPPER** = False

**RECURSIVE\_DEFAULT** = True

**adopt\_declaration** (*decl*)

**attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

**cache**

Implementation detail.

Reference to instance of *algorithms\_cache\_t* class.

**calldef** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to “calldef” declaration, that is matched defined criteria

**calldefs** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of *calldef\_t* declarations, that are matched defined criteria

**casting\_operator** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to casting operator declaration, that is matched defined criteria

**casting\_operators** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of casting operator declarations, that are matched defined criteria

**class\_** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to class declaration, that is matched defined criteria



**classes** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)  
 returns a set of class declarations, that are matched defined criteria

**clear\_optimizer** ()  
 Cleans query optimizer state

**constructor** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)  
 returns reference to constructor declaration, that is matched defined criteria

**constructors** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)  
 returns a set of constructor declarations, that are matched defined criteria

**create\_decl\_string** (*with\_defaults=True*)

**decl** (*name=None, function=None, decl\_type=None, header\_dir=None, header\_file=None, recursive=None*)  
 returns reference to declaration, that is matched defined criteria

**decl\_string**  
 Declaration full name.

**declaration\_not\_found\_t**

**declarations**  
 List of children declarations.  
**Returns** list[declaration\_t]

**decls** (*name=None, function=None, decl\_type=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)  
 returns a set of declarations, that are matched defined criteria

**decorated\_name**  
 Unique declaration name extracted from a binary file ( .map, .dll, .so, etc ).  
 @type: str

**demangled**  
 Declaration name, reconstructed from GCCXML generated unique name.  
 @type: str

**enum** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None*)  
 Deprecated method. Use the enumeration() method instead.  
 Deprecated since v1.9.0. Will be removed in v2.0.0

**enumeration** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None*)  
 returns reference to enumeration declaration, that is matched defined criteria

**enumerations** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)  
 returns a set of enumeration declarations, that are matched defined criteria

**enums** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)  
 Deprecated method. Use the enumerations() method instead.  
 Deprecated since v1.9.0. Will be removed in v2.0.0

**free\_fun** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)  
 Deprecated method. Use the free\_function() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**free\_function** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

Returns reference to free function declaration that matches a defined criteria.

**free\_functions** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

Returns a set of free function declarations that match a defined criteria.

**free\_funs** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

Deprecated method. Use the free\_functions() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**free\_operator** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

Returns reference to free operator declaration that matches a defined criteria.

**free\_operators** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

Returns a set of free operator declarations that match a defined criteria.

**get\_mangled\_name** ()

**i\_depend\_on\_them** (*recursive=True*)

**init\_optimizer** ()

Initializes query optimizer state.

**There are 4 internals hash tables:**

1. from type to declarations
2. from type to declarations for non-recursive queries
3. from type to name to declarations
4. from type to name to declarations for non-recursive queries

Almost every query includes declaration type information. Also very common query is to search some declaration(s) by name or full name. Those hash tables allows to search declaration very quick.

**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: location\_t

**mangled**

Unique declaration name generated by the compiler.

For GCCXML, you can get the mangled name for all the declarations. When using CastXML, calling mangled is only allowed on functions and variables. For other declarations it will raise an exception.

**Returns** the mangled name

**Return type** str

**mem\_fun** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

Deprecated method. Use the member\_function() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**mem\_funs** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

Deprecated method. Use the member\_functions() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**mem\_oper** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

Deprecated method. Use the member\_operator() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**mem\_opsers** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

Deprecated method. Use the member\_operators() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**member\_function** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to member declaration, that is matched defined criteria

**member\_functions** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of member function declarations, that are matched defined criteria

**member\_operator** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to member operator declaration, that is matched defined criteria

**member\_operators** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of member operator declarations, that are matched defined criteria

**multiple\_declarations\_found\_t**

**name**

Declaration name @type: str

**namespace** (*name=None, function=None, recursive=None*)

Returns reference to namespace declaration that matches a defined criteria.

**namespaces** (*name=None, function=None, recursive=None, allow\_empty=None*)

Returns a set of namespace declarations that match a defined criteria.

**nss** (*name=None, function=None, recursive=None, allow\_empty=None*)

Deprecated method. Use the namespaces() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**operator** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to operator declaration, that is matched defined criteria

**operators** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of operator declarations, that are matched defined criteria

**parent**

Reference to parent declaration.

@type: declaration\_t

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**remove\_declaration** (*decl*)

Removes declaration from members list.

**Parameters** **decl** (*declaration\_t*) – declaration to be removed

**take\_parenting** (*inst*)

Takes parenting from inst and transfers it to self.

**Parameters** **inst** (*namespace\_t*) – a namespace declaration

**top\_parent**

Reference to top parent declaration.

@type: *declaration\_t*

**typedef** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to typedef declaration, that is matched defined criteria

**typedefs** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of typedef declarations, that are matched defined criteria

**variable** (*name=None, function=None, decl\_type=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to variable declaration, that is matched defined criteria

**variables** (*name=None, function=None, decl\_type=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of variable declarations, that are matched defined criteria

**pygccxml.declarations.pattern\_parser module**

Implementation details

**class parser\_t** (*pattern\_char\_begin, pattern\_char\_end, pattern\_char\_separator*)

Bases: *object*

implementation details

**NOT\_FOUND** = (-1, -1)

implementation details

**args** (*decl\_string*)

Extracts a list of arguments from the provided declaration string.

Implementation detail. Example usages: Input: `myClass<std::vector<int>, std::vector<double>>` Output: `[std::vector<int>, std::vector<double>]`

**Parameters** **decl\_string** (*str*) – the full declaration string

**Returns** list of arguments as strings

**Return type** list

**find\_args** (*text, start=None*)

implementation details

**has\_pattern** (*decl\_string*)

Implementation detail

**join** (*name, args, arg\_separator=None*)

implementation details

**name** (*decl\_string*)

implementation details

**normalize** (*decl\_string, arg\_separator=None*)

implementation details

**split** (*decl\_string*)

implementation details

**split\_recursive** (*decl\_string*)

implementation details

### pygccxml.declarations.runtime\_errors module

**exception declaration\_not\_found\_t** (*decl\_matcher*)

Bases: `exceptions.RuntimeError`

Exception raised when the declaration could not be found

**args**

**message**

**exception multiple\_declarations\_found\_t** (*decl\_matcher*)

Bases: `exceptions.RuntimeError`

Exception raised when more than one declaration was found

**args**

**message**

**exception visit\_function\_has\_not\_been\_found\_t** (*visitor, decl\_inst*)

Bases: `exceptions.RuntimeError`

Exception that is raised, from `apply_visitor()`, when a visitor could not be applied.

**args**

**message**

### pygccxml.declarations.scopedef module

Defines *scopedef\_t* class

**declaration\_files** (*decl\_or\_decls*)

Returns set of files

Every declaration is declared in some file. This function returns set, that contains all file names of declarations.

**Parameters** **decl\_or\_decls** (`declaration_t` or [`declaration_t`]) – reference to list of declaration's or single declaration

**Return type** `set(declaration file names)`

**find\_all\_declarations** (*declarations*, *decl\_type=None*, *name=None*, *parent=None*, *recursive=True*, *fullname=None*)

Returns a list of all declarations that match criteria, defined by developer.

For more information about arguments see `match_declaration_t` class.

**Return type** [ matched declarations ]

**find\_declaration** (*declarations*, *decl\_type=None*, *name=None*, *parent=None*, *recursive=True*, *fullname=None*)

Returns single declaration that match criteria, defined by developer. If more the one declaration was found None will be returned.

For more information about arguments see `match_declaration_t` class.

**Return type** matched declaration `declaration_t` or None

**find\_first\_declaration** (*declarations*, *decl\_type=None*, *name=None*, *parent=None*, *recursive=True*, *fullname=None*)

Returns first declaration that match criteria, defined by developer.

For more information about arguments see `match_declaration_t` class.

**Return type** matched declaration `declaration_t` or None

**make\_flatten** (*decl\_or\_decls*)

Converts tree representation of declarations to flatten one.

**Parameters** **decl\_or\_decls** (`declaration_t` or [ `declaration_t` ]) – reference to list of declaration's or single declaration

**Return type** [ all internal declarations ]

**class matcher**

Bases: `object`

Class-namespace, contains implementation of a few “find” algorithms

**static find** (*decl\_matcher*, *decls*, *recursive=True*)

Returns a list of declarations that match *decl\_matcher* defined criteria or None

**Parameters**

- **decl\_matcher** – Python callable object, that takes one argument - reference to a declaration
- **decls** – the search scope, `:class:declaration_t` object or `:class:declaration_t` objects list
- **recursive** – boolean, if True, the method will run *decl\_matcher* on the internal declarations too

**static find\_single** (*decl\_matcher*, *decls*, *recursive=True*)

Returns a reference to the declaration, that match *decl\_matcher* defined criteria.

if a unique declaration could not be found the method will return None.

**Parameters**

- **decl\_matcher** – Python callable object, that takes one argument - reference to a declaration
- **decls** – the search scope, `:class:declaration_t` object or `:class:declaration_t` objects list
- **recursive** – boolean, if True, the method will run *decl\_matcher* on the internal declarations too

**static** `get_single` (*decl\_matcher*, *decls*, *recursive=True*)

Returns a reference to declaration, that match *decl\_matcher* defined criteria.

If a unique declaration could not be found, an appropriate exception will be raised.

#### Parameters

- **decl\_matcher** – Python callable object, that takes one argument - reference to a declaration
- **decls** – the search scope, :class:declaration\_t object or :class:declaration\_t objects list
- **recursive** – boolean, if True, the method will run *decl\_matcher* on the internal declarations too

**class** `scopedef_t` (*name=''*)

Bases: `pygccxml.declarations.declaration.declaration_t`

Base class for `namespace_t` and `class_t` classes.

This is the base class for all declaration classes that may have children nodes. The children can be accessed via the `scopedef_t.declarations` property.

Also this class provides “get/select/find” interface. Using this class you can get instance or instances of internal declaration(s).

You can find declaration(s) using next criteria:

- 1.*name* - declaration name, could be full qualified name
- 2.**header\_dir - directory, to which belongs file, that the** declaration was declared in. *header\_dir* should be absolute path.
- 3.*header\_file* - file that the declaration was declared in.
- 4.**function - user ( your ) custom criteria. The interesting thing** is that this function will be joined with other arguments (criteria).
- 5.**recursive - the search declaration range, if True will be search** in internal declarations too.

Every “”query”” API, takes name or function as the first argument.

```
global_namespace.member_function("do_something")
```

the statement returns reference to member function named “do\_something”. If there the function doesn’t exist or more than one function exists, an exception is raised.

If you want to query for many declarations, use other function(s):

```
do_something = global_namespace.member_functions("do_something")
```

the statement returns `mdecl_wrapper_t` instance. That object will save you writing *for* loops. For more information see the `class` documentation.

**ALLOW\_EMPTY\_MDECL\_WRAPPER = False**

**RECURSIVE\_DEFAULT = True**

#### attributes

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

#### cache

Implementation detail.

Reference to instance of `algorithms_cache_t` class.

**calldef** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to “calldef” declaration, that is matched defined criteria

**calldefs** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of `calldef_t` declarations, that are matched defined criteria

**casting\_operator** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to casting operator declaration, that is matched defined criteria

**casting\_operators** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of casting operator declarations, that are matched defined criteria

**class\_** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to class declaration, that is matched defined criteria

**classes** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of class declarations, that are matched defined criteria

**clear\_optimizer** ()

Cleans query optimizer state

**constructor** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to constructor declaration, that is matched defined criteria

**constructors** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of constructor declarations, that are matched defined criteria

**create\_decl\_string** (*with\_defaults=True*)

**decl** (*name=None, function=None, decl\_type=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to declaration, that is matched defined criteria

**decl\_string**

Declaration full name.

**declaration\_not\_found\_t**

**declarations**

List of children declarations.

**Returns** List[declarations.declaration\_t]

**decls** (*name=None, function=None, decl\_type=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of declarations, that are matched defined criteria

**decorated\_name**

Unique declaration name extracted from a binary file ( .map, .dll, .so, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str



**enum** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None*)

Deprecated method. Use the enumeration() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**enumeration** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to enumeration declaration, that is matched defined criteria

**enumerations** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of enumeration declarations, that are matched defined criteria

**enums** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

Deprecated method. Use the enumerations() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**get\_mangled\_name** ()

**i\_depend\_on\_them** (*recursive=True*)

**init\_optimizer** ()

Initializes query optimizer state.

**There are 4 internals hash tables:**

1. from type to declarations
2. from type to declarations for non-recursive queries
3. from type to name to declarations
4. from type to name to declarations for non-recursive queries

Almost every query includes declaration type information. Also very common query is to search some declaration(s) by name or full name. Those hash tables allows to search declaration very quick.

**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: location\_t

**mangled**

Unique declaration name generated by the compiler.

For GCCXML, you can get the mangled name for all the declarations. When using CastXML, calling mangled is only allowed on functions and variables. For other declarations it will raise an exception.

**Returns** the mangled name

**Return type** str

**mem\_fun** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

Deprecated method. Use the member\_function() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**mem\_funcs** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

Deprecated method. Use the member\_functions() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**mem\_oper** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

Deprecated method. Use the member\_operator() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**mem\_operators** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

Deprecated method. Use the member\_operators() method instead.

Deprecated since v1.9.0. Will be removed in v2.0.0

**member\_function** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to member declaration, that is matched defined criteria

**member\_functions** (*name=None, function=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of member function declarations, that are matched defined criteria

**member\_operator** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to member operator declaration, that is matched defined criteria

**member\_operators** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of member operator declarations, that are matched defined criteria

#### **multiple\_declarations\_found\_t**

##### **name**

Declaration name @type: str

**operator** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to operator declaration, that is matched defined criteria

**operators** (*name=None, function=None, symbol=None, return\_type=None, arg\_types=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of operator declarations, that are matched defined criteria

##### **parent**

Reference to parent declaration.

@type: declaration\_t

##### **partial\_decl\_string**

Declaration full name.

##### **partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

##### **remove\_declaration** (*decl*)

##### **top\_parent**

Reference to top parent declaration.

@type: declaration\_t

**typedef** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to typedef declaration, that is matched defined criteria

**typedefs** (*name=None, function=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of typedef declarations, that are matched defined criteria

**variable** (*name=None, function=None, decl\_type=None, header\_dir=None, header\_file=None, recursive=None*)

returns reference to variable declaration, that is matched defined criteria

**variables** (*name=None, function=None, decl\_type=None, header\_dir=None, header\_file=None, recursive=None, allow\_empty=None*)

returns a set of variable declarations, that are matched defined criteria

## pygccxml.declarations.smart\_pointer\_traits module

**class auto\_ptr\_traits**

Bases: object

implements functionality, needed for convenient work with *std::auto\_ptr* pointers

**static is\_smart\_pointer** (*type\_*)

returns True, if type represents instantiation of *boost::shared\_ptr*, False otherwise

**static value\_type** (*type\_*)

returns reference to *boost::shared\_ptr* value type

**class internal\_type\_traits**

Bases: object

small convenience class, which provides access to internal types

**static get\_by\_name** (*type\_, name*)

**class smart\_pointer\_traits**

Bases: object

implements functionality, needed for convenient work with smart pointers

**static is\_smart\_pointer** (*type\_*)

returns True, if type represents instantiation of *boost::shared\_ptr* or *std::shared\_ptr*, False otherwise

**static value\_type** (*type\_*)

returns reference to *boost::shared\_ptr* or *std::shared\_ptr* value type

## pygccxml.declarations.templates module

template instantiation parser

This module provides functionality necessary to

- `parse`
- `split`
- `join`
- `normalize`

C++ template instantiations

**args** (*decl\_string*)

returns list of template arguments

**Return type** [*str*]

**is\_instantiation** (*decl\_string*)

returns True if *decl\_string* is template instantiation and False otherwise

**Parameters** **decl\_string** (*str*) – string that should be checked for pattern presence

**Return type** bool

**join** (*name\_*, *args\_*)

returns name< argument\_1, argument\_2, ..., argument\_n >

**name** (*decl\_string*)

returns name of instantiated template

**Return type** str

**normalize** (*decl\_string*)

returns *decl\_string*, which contains “normalized” spaces

this functionality allows to implement comparison of 2 different string which are actually same: x::y< z > and x::y<z>

**normalize\_full\_name\_false** (*decl*)

Cached variant of normalize

**Parameters** **decl** (*declaration.declaration\_t*) – the declaration

**Returns** normalized name

**Return type** str

**normalize\_full\_name\_true** (*decl*)

Cached variant of normalize

**Parameters** **decl** (*declaration.declaration\_t*) – the declaration

**Returns** normalized name

**Return type** str

**normalize\_name** (*decl*)

Cached variant of normalize

**Parameters** **decl** (*declaration.declaration\_t*) – the declaration

**Returns** normalized name

**Return type** str

**normalize\_partial\_name** (*decl*)

Cached variant of normalize

**Parameters** **decl** (*declaration.declaration\_t*) – the declaration

**Returns** normalized name

**Return type** str

**split** (*decl\_string*)

returns (name, [arguments] )

**split\_recursive** (*decl\_string*)

returns [(name, [arguments])]

## pygccxml.declarations.traits\_impl\_details module

**class impl\_details**

Bases: object

implementation details

**static find\_value\_type** (*global\_ns, value\_type\_str*)  
implementation details

**static is\_defined\_in\_XXX** (*xxx, cls*)  
Small helper method that checks whether the class *cls* is defined under *::xxx* namespace

## pygccxml.declarations.type\_traits module

defines few algorithms, that deals with different C++ type properties

Are you aware of `boost::type_traits` library? pygccxml implements the same functionality.

This module contains a set of very specific traits functions/classes, each of which encapsulate a single trait from the C++ type system. For example: \* is a type a pointer or a reference type ? \* does a type have a trivial constructor ? \* does a type have a const-qualifier ?

**array\_item\_type** (*type\_*)  
returns array item type

**array\_size** (*type\_*)  
returns array size

**base\_type** (*type\_*)  
returns base type.

For *const int* will return *int*

**decompose\_class** (*type\_*)  
implementation details

**decompose\_type** (*tp*)  
Implementation detail

**does\_match\_definition** (*given, main, secondary*)  
implementation details

**is\_arithmetic** (*type\_*)  
returns True, if type represents C++ integral or floating point type, False otherwise

**is\_array** (*type\_*)  
returns True, if type represents C++ array type, False otherwise

**is\_bool** (*type\_*)  
Check if type is of boolean type.

**Parameters** *type* (*type\_t*) – The type to be checked

**Returns** True if type is a boolean, False otherwise.

**Return type** bool

**is\_calldef\_pointer** (*type\_*)  
returns True, if type represents pointer to free/member function, False otherwise

**is\_const** (*type\_*)  
returns True, if type represents C++ const type, False otherwise

**is\_elaborated** (*type\_*)  
returns True, if type represents C++ elaborated type, False otherwise

**is\_floating\_point** (*type\_*)  
returns True, if type represents C++ floating point type, False otherwise

**is\_fundamental** (*type\_*)  
returns True, if type represents C++ fundamental type

**is\_integral** (*type\_*)  
Check if type is a C++ integral type

**Parameters** **type** (*type\_t*) – The type to be checked

**Returns** True if type is a C++ integral type, False otherwise.

**Return type** bool

**is\_pointer** (*type\_*)  
returns True, if type represents C++ pointer type, False otherwise

**is\_reference** (*type\_*)  
returns True, if type represents C++ reference type, False otherwise

**is\_same** (*type1*, *type2*)  
returns True, if type1 and type2 are same types

**is\_std\_ostream** (*type\_*)  
Returns True, if type represents C++ `std::ostream`, False otherwise.

**is\_std\_string** (*type\_*)  
Returns True, if type represents C++ `std::string`, False otherwise.

**is\_std\_wostream** (*type\_*)  
Returns True, if type represents C++ `std::wostream`, False otherwise.

**is\_std\_wstring** (*type\_*)  
Returns True, if type represents C++ `std::wstring`, False otherwise.

**is\_void** (*type\_*)  
Check if type is of void type.

**Parameters** **type** (*type\_t*) – The type to be checked

**Returns** True if type is void, False otherwise.

**Return type** bool

**is\_void\_pointer** (*type\_*)  
returns True, if type represents `void*`, False otherwise

**is\_volatile** (*type\_*)  
returns True, if type represents C++ volatile type, False otherwise

**remove\_alias** (*type\_*)  
Returns *type\_t* without typedef

**Parameters** **type** (*type\_t* | *declaration\_t*) – type or declaration

**Returns** the type associated to the inputted declaration

**Return type** *type\_t*

**remove\_const** (*type\_*)  
removes const from the type definition

If type is not const type, it will be returned as is

**remove\_cv** (*type\_*)

removes const and volatile from the type definition

**remove\_declarated** (*type\_*)

removes type-declaration class-binder *declarated\_t* from the *type\_*

If *type\_* is not *declarated\_t*, it will be returned as is

**remove\_elaborated** (*type\_*)

removes type-declaration class-binder *elaborated\_t* from the *type\_*

If *type\_* is not *elaborated\_t*, it will be returned as is

**remove\_pointer** (*type\_*)

removes pointer from the type definition

If type is not pointer type, it will be returned as is.

**remove\_reference** (*type\_*)

removes reference from the type definition

If type is not reference type, it will be returned as is.

**remove\_volatile** (*type\_*)

removes volatile from the type definition

If type is not volatile type, it will be returned as is

## pygccxml.declarations.type\_traits\_classes module

**class\_declaration\_traits** = <pygccxml.declarations.type\_traits\_classes.declaration\_xxx\_traits object>

implements functionality, needed for convenient work with C++ class declarations

**class\_traits** = <pygccxml.declarations.type\_traits\_classes.declaration\_xxx\_traits object>

implements functionality, needed for convenient work with C++ classes

**class\_declaration\_xxx\_traits** (*declaration\_class*)

Bases: object

this class implements the functionality needed for convenient work with declaration classes

### Implemented functionality:

- find out whether a declaration is a desired one
- get reference to the declaration

**get\_declaration** (*type\_*)

returns reference to the declaration

Precondition: *self.is\_my\_case( type )* == True

**is\_my\_case** (*type\_*)

returns True, if type represents the desired declaration, False otherwise

**enum\_declaration** = <bound method *declaration\_xxx\_traits.get\_declaration* of <pygccxml.declarations.type\_traits\_classes.*...*>

returns reference to enum declaration

**enum\_traits** = <pygccxml.declarations.type\_traits\_classes.declaration\_xxx\_traits object>

implements functionality, needed for convenient work with C++ enums

**find\_copy\_constructor** (*type\_*)

Returns reference to copy constructor.

**Parameters** **type** (*declarations.class\_t*) – the class to be searched.

**Returns** the copy constructor

**Return type** *declarations.constructor\_t*

**find\_noncopyable\_vars** (*class\_type, already\_visited\_cls\_vars=None*)

Returns list of all *noncopyable* variables.

If an *already\_visited\_cls\_vars* list is provided as argument, the returned list will not contain these variables. This list will be extended with whatever variables pointing to classes have been found.

**Parameters**

- **class\_type** (*declarations.class\_t*) – the class to be searched.
- **already\_visited\_cls\_vars** (*list*) – optional list of vars that should not be checked a second time, to prevent infinite recursions.

**Returns** list of all *noncopyable* variables.

**Return type** *list*

**find\_trivial\_constructor** (*type\_*)

Returns reference to trivial constructor.

**Parameters** **type** (*declarations.class\_t*) – the class to be searched.

**Returns** the trivial constructor

**Return type** *declarations.constructor\_t*

**has\_any\_non\_copyconstructor** (*decl\_type*)

if class has any public constructor, which is not copy constructor, this function will return list of them, otherwise None

**has\_copy\_constructor** (*class\_*)

if class has public copy constructor, this function will return reference to it, None otherwise

**has\_destructor** (*class\_*)

if class has destructor, this function will return reference to it, None otherwise

**has\_public\_assign** (*class\_*)

returns True, if class has public assign operator, False otherwise

**has\_public\_constructor** (*class\_*)

if class has any public constructor, this function will return list of them, otherwise None

**has\_public\_destructor** (*decl\_type*)

returns True, if class has public destructor, False otherwise

**has\_trivial\_constructor** (*class\_*)

if class has public trivial constructor, this function will return reference to it, None otherwise

**has\_vtable** (*decl\_type*)

True, if class has virtual table, False otherwise

**is\_base\_and\_derived** (*based, derived*)

returns True, if there is “base and derived” relationship between classes, False otherwise

**is\_binary\_operator** (*oper*)

returns True, if operator is binary operator, otherwise False



**is\_class** = <bound method declaration\_xxx\_traits.is\_my\_case of <pygccxml.declarations.type\_traits\_classes.declaration\_xxx...  
 returns True, if type represents C++ class definition, False otherwise

**is\_class\_declaration** = <bound method declaration\_xxx\_traits.is\_my\_case of <pygccxml.declarations.type\_traits\_classes...  
 returns True, if type represents C++ class declaration, False otherwise

**is\_convertible** (*source*, *target*)  
 returns True, if source could be converted to target, otherwise False

**is\_copy\_constructor** (*constructor*)  
 Check if the declaration is a copy constructor,

**Parameters** **constructor** (*declarations.constructor\_t*) – the constructor to be checked.

**Returns** True if this is a copy constructor, False instead.

**Return type** bool

**is\_enum** = <bound method declaration\_xxx\_traits.is\_my\_case of <pygccxml.declarations.type\_traits\_classes.declaration\_xxx...  
 returns True, if type represents C++ enumeration declaration, False otherwise

**is\_noncopyable** (*class\_*, *already\_visited\_cls\_vars*=None)  
 Checks if class is non copyable

**Parameters**

- **class** (*declarations.class\_t*) – the class to be checked
- **already\_visited\_cls\_vars** (*list*) – optional list of vars that should not be checked a second time, to prevent infinite recursions. In general you can ignore this argument, it is mainly used during recursive calls of `is_noncopyable()` done by pygccxml.

**Returns** if the class is non copyable

**Return type** bool

**is\_struct** (*declaration*)  
 Returns True if declaration represents a C++ struct

**Parameters** **declaration** (*declaration\_t*) – the declaration to be checked.

**Returns** True if declaration represents a C++ struct

**Return type** bool

**is\_trivial\_constructor** (*constructor*)  
 Check if the declaration is a trivial constructor.

**Parameters** **constructor** (*declarations.constructor\_t*) – the constructor to be checked.

**Returns** True if this is a trivial constructor, False instead.

**Return type** bool

**is\_unary\_operator** (*oper*)  
 returns True, if operator is unary operator, otherwise False

**is\_union** (*declaration*)  
 Returns True if declaration represents a C++ union

**Parameters** **declaration** (*declaration\_t*) – the declaration to be checked.

**Returns** True if declaration represents a C++ union

**Return type** bool

**pygccxml.declarations.type\_visitor module**

defines types visitor class interface

**class** `type_visitor_t`

Bases: `object`

types visitor interface

All functions within this class should be redefined in derived classes.

**visit\_array**()

**visit\_bool**()

**visit\_char**()

**visit\_complex\_double**()

**visit\_complex\_float**()

**visit\_complex\_long\_double**()

**visit\_const**()

**visit\_declarated**()

**visit\_double**()

**visit\_elaborated**()

**visit\_ellipsis**()

**visit\_float**()

**visit\_free\_function\_type**()

**visit\_int**()

**visit\_int128**()

**visit\_jboolean**()

**visit\_jbyte**()

**visit\_jchar**()

**visit\_jdouble**()

**visit\_jfloat**()

**visit\_jint**()

**visit\_jlong**()

**visit\_jshort**()

**visit\_long\_double**()

**visit\_long\_int**()

**visit\_long\_long\_int**()

**visit\_long\_long\_unsigned\_int**()

**visit\_long\_unsigned\_int**()

**visit\_member\_function\_type**()

**visit\_member\_variable\_type**()

```

visit_pointer()
visit_reference()
visit_restrict()
visit_short_int()
visit_short_unsigned_int()
visit_signed_char()
visit_uint128()
visit_unsigned_char()
visit_unsigned_int()
visit_void()
visit_volatile()
visit_wchar()

```

### pygccxml.declarations.typedef module

defines class that describes C++ typedef declaration

```

class typedef_t (name='', decl_type=None)
    Bases: pygccxml.declarations.declaration.declaration_t, pygccxml.declarations.byte_info.byte_info

```

describes C++ typedef declaration

#### **attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

#### **byte\_align**

Alignment of this declaration/type in bytes

**Returns** Alignment of this declaration/type in bytes

**Return type** int

#### **byte\_size**

Size of this declaration/type in bytes

**Returns** Size of this declaration/type in bytes

**Return type** int

#### **cache**

Implementation detail.

Reference to instance of `algorithms_cache_t` class.

**create\_decl\_string** (*with\_defaults=True*)

#### **decl\_string**

Declaration full name.

#### **decl\_type**

reference to the original `decl_type`

**decorated\_name**

Unique declaration name extracted from a binary file ( .map, .dll, .so, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**get\_mangled\_name()****i\_depend\_on\_them** (*recursive=True*)**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: location\_t

**mangled**

Unique declaration name generated by the compiler.

For GCCXML, you can get the mangled name for all the declarations. When using CastXML, calling mangled is only allowed on functions and variables. For other declarations it will raise an exception.

**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**parent**

Reference to parent declaration.

@type: declaration\_t

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**top\_parent**

Reference to top parent declaration.

@type: declaration\_t

## pygccxml.declarations.variable module

defines class that describes C++ global and member variable declaration

**class variable\_t** (*name='', decl\_type=None, type\_qualifiers=None, value=None, bits=None, mangled=None*)

Bases: `pygccxml.declarations.declaration.declaration_t`

describes C++ global and member variable declaration

**access\_type**

**attributes**

GCCXML attributes, set using `__attribute__((gccxml("...")))`

@type: str

**bits**

integer, that contains information about how many bit takes bit field

**byte\_offset**

integer, offset of the field from the beginning of class.

**cache**

Implementation detail.

Reference to instance of `algorithms_cache_t` class.

**create\_decl\_string** (*with\_defaults=True*)

**decl\_string**

Declaration full name.

**decl\_type**

reference to the variable `decl_type`

**decorated\_name**

Unique declaration name extracted from a binary file ( `.map`, `.dll`, `.so`, etc ).

@type: str

**demangled**

Declaration name, reconstructed from GCCXML generated unique name.

@type: str

**get\_mangled\_name** ()

**i\_depend\_on\_them** (*recursive=True*)

**is\_artificial**

Describes whether declaration is compiler generated or not

@type: bool

**location**

Location of the declaration within source file

@type: `location_t`

**mangled**

Unique declaration name generated by the compiler.

**Returns** the mangled name

**Return type** str

**name**

Declaration name @type: str

**parent**

Reference to parent declaration.

@type: `declaration_t`

**partial\_decl\_string**

Declaration full name.

**partial\_name**

Declaration name, without template default arguments.

Right now std containers is the only classes that support this functionality.

**top\_parent**

Reference to top parent declaration.

@type: declaration\_t

**type\_qualifiers**

reference to the type\_qualifiers\_t instance

**value**

string, that contains the variable value

## pygccxml.declarations.xml\_generators module

Contains enumeration of all xml\_generators supported by the project.

**on\_missing\_functionality** (xml\_generator, functionality)

## pygccxml.parser package

Parser sub-package.

**parse** (files, config=None, compilation\_mode='file by file', cache=None)

Parse header files.

**Parameters**

- **files** (list of str) – The header files that should be parsed
- **config** (parser.xml\_generator\_configuration\_t) – Configuration object or None
- **compilation\_mode** (parser.COMPILATION\_MODE) – Determines whether the files are parsed individually or as one single chunk
- **cache** (parser.cache\_base\_t or str) – Declaration cache (None=no cache)

**Return type** list of declarations.declaration\_t

**parse\_string** (content, config=None)

**parse\_xml\_file** (content, config=None)

## Submodules

### pygccxml.parser.config module

Defines C++ parser configuration classes.

**create\_compiler\_path** (xml\_generator, compiler\_path)

Try to guess a path for the compiler.

If you want to use a specific compiler, please provide the compiler path manually, as the guess may not be what you are expecting. Providing the path can be done by passing it as an argument (compiler\_path) to the xml\_generator\_configuration\_t() or by defining it in your pygccxml configuration file.

**load\_xml\_generator\_configuration** (*configuration*, *\*\*defaults*)

Loads CastXML or GCC-XML configuration.

**Parameters**

- **configuration** (*string/configparser.ConfigParser*) – can be a string (file path to a configuration file) or instance of `configparser.ConfigParser`.
- **defaults** – can be used to override single configuration values.

**Returns** a configuration object

**Return type** `xml_generator_configuration_t`

The file passed needs to be in a format that can be parsed by `configparser.ConfigParser`.

An example configuration file skeleton can be found [here](#).

```
class parser_configuration_t (working_directory='.', include_paths=None, define_symbols=None,
                             undefine_symbols=None, cflags='', compiler=None,
                             xml_generator=None, keep_xml=False, compiler_path=None,
                             flags=None, castxml_epic_version=None)
```

Bases: `object`

C++ parser configuration holder

This class serves as a base class for the parameters that can be used to customize the call to a C++ parser.

This class also allows users to work with relative files paths. In this case files are searched in the following order:

- 1.current directory
- 2.working directory
- 3.additional include paths specified by the user

**append\_cflags** (*val*)

**castxml\_epic\_version**

File format version used by castxml.

**cflags**

additional flags to pass to compiler

**clone** ()

**compiler**

get compiler name to simulate

**compiler\_path**

Get the path for the compiler.

**define\_symbols**

list of “define” directives

**flags**

Optional flags for pygccxml.

**include\_paths**

list of include paths to look for header files

**keep\_xml**

Are xml files kept after errors.

**raise\_on\_wrong\_settings** ()

Validates the configuration settings and raises `RuntimeError` on error

**undefine\_symbols**  
list of “undefine” directives

**working\_directory**

**xml\_generator**  
get xml\_generator (gccxml or castxml)

```
class xml_generator_configuration_t (gccxml_path='',      xml_generator_path='',      work-
                                   ing_directory='',      include_paths=None,      de-
                                   fine_symbols=None,      undefine_symbols=None,
                                   start_with_declarations=None,      ig-
                                   nore_gccxml_output=False,      cflags='',
                                   compiler=None,      xml_generator=None,
                                   keep_xml=False,      compiler_path=None,      flags=None,
                                   castxml_epic_version=None)
```

Bases: `pygccxml.parser.config.parser_configuration_t`

Configuration object to collect parameters for invoking gccxml or castxml.

This class serves as a container for the parameters that can be used to customize the call to gccxml or castxml.

**append\_cflags** (*val*)

**castxml\_epic\_version**  
File format version used by castxml.

**cflags**  
additional flags to pass to compiler

**clone** ()

**compiler**  
get compiler name to simulate

**compiler\_path**  
Get the path for the compiler.

**define\_symbols**  
list of “define” directives

**flags**  
Optional flags for pygccxml.

**ignore\_gccxml\_output**  
set this property to True, if you want pygccxml to ignore any error warning that comes from gccxml

**include\_paths**  
list of include paths to look for header files

**keep\_xml**  
Are xml files kept after errors.

**raise\_on\_wrong\_settings** ()

**start\_with\_declarations**  
list of declarations gccxml should start with, when it dumps declaration tree

**undefine\_symbols**  
list of “undefine” directives

**working\_directory**

**xml\_generator**  
get xml\_generator (gccxml or castxml)



**xml\_generator\_from\_xml\_file**

Configuration object containing information about the xml generator read from the xml file.

**Returns** configuration object

**Return type** *utils.xml\_generators*

**xml\_generator\_path**

XML generator binary location

**pygccxml.parser.declarations\_cache module****class cache\_base\_t**

Bases: `object`

**cached\_value** (*source\_file, configuration*)

Return declarations, we have cached, for the *source\_file* and the given *configuration*.

**Parameters**

- **source\_file** – path to the C++ source file being parsed.
- **configuration** – configuration that was used for parsing.

**flush** ()

Flush (write out) the cache to disk if needed.

**logger** = <logging.Logger object>**update** (*source\_file, configuration, declarations, included\_files*)

update cache entry

**Parameters**

- **source\_file** – path to the C++ source file being parsed
- **configuration** – configuration used in parsing  
`xml_generator_configuration_t`
- **declarations** – declaration tree found when parsing
- **included\_files** – files included by parsing.

**configuration\_signature** (*config*)

Return a signature for a configuration (`xml_generator_configuration_t`) object.

This can then be used as a key in the cache. This method must take into account anything about a configuration that could cause the declarations generated to be different between runs.

**class dummy\_cache\_t**

Bases: *pygccxml.parser.declarations\_cache.cache\_base\_t*

This is an empty cache object.

By default no caching is enabled in pygccxml.

**cached\_value** (*source\_file, configuration*)**flush** ()**logger** = <logging.Logger object>**update** (*source\_file, configuration, declarations, included\_files*)

**class file\_cache\_t** (*name*)

Bases: `pygccxml.parser.declarations_cache.cache_base_t`

Cache implementation to store data in a pickled form in a file. This class contains some cache logic that keeps track of which entries have been ‘hit’ in the cache and if an entry has not been hit then it is deleted at the time of the flush(). This keeps the cache from growing larger when files change and are not used again.

**cached\_value** (*source\_file, configuration*)

Attempt to lookup the cached declarations for the given file and configuration.

Returns None if declaration not found or signature check fails.

**flush** ()

**logger** = <logging.Logger object>

**update** (*source\_file, configuration, declarations, included\_files*)

Update a cached record with the current key and value contents.

**file\_signature** (*filename*)

Return a signature for a file.

**class record\_t** (*source\_signature, config\_signature, included\_files, included\_files\_signature, declarations*)

Bases: `object`

**config\_signature**

**static create\_key** (*source\_file, configuration*)

**declarations**

**included\_files**

**included\_files\_signature**

**key** ()

**source\_signature**

**was\_hit**

## pygccxml.parser.declarations\_joiner module

**bind\_aliases** (*decls*)

This function binds between class and it’s typedefs.

**Parameters** **decls** – list of all declarations

**Return type** None

**join\_declarations** (*namespace*)

## pygccxml.parser.directory\_cache module

directory cache implementation.

This module contains the implementation of a cache that uses individual files, stored in a dedicated cache directory, to store the cached contents.

The `parser.directory_cache_t` class instance could be passed as the *cache* argument of the `parser.parse()` function.

**class** `directory_cache_t` (*dir='cache', directory='cache', compression=False, sha1\_sigs=True*)

Bases: `pygccxml.parser.declarations_cache.cache_base_t`

cache class that stores its data as multiple files inside a directory.

The cache stores one index file called *index.dat* which is always read by the cache when the cache object is created. Each header file will have its corresponding *.cache* file that stores the declarations found in the header file. The index file is used to determine whether a *.cache* file is still valid or not (by checking if one of the dependent files (i.e. the header file itself and all included files) have been modified since the last run).

**cached\_value** (*source\_file, configuration*)

Return the cached declarations or None.

#### Parameters

- **source\_file** (*str*) – Header file name
- **configuration** (`parser.xml_generator_configuration_t`) – Configuration object

**Return type** Cached declarations or None

**flush** ()

Save the index table to disk.

**logger** = `<logging.Logger object>`

**update** (*source\_file, configuration, declarations, included\_files*)

Replace a cache entry by a new value.

#### Parameters

- **source\_file** (*str*) – a C++ source file name.
- **configuration** (`xml_generator_configuration_t`) – configuration object.
- **declarations** (*pickable object*) – declarations contained in the *source\_file*
- **included\_files** (*list of str*) – included files

**class** `filename_entry_t` (*filename*)

Bases: `object`

This is a record stored in the `filename_repository_t` class.

The class is an internal class used in the implementation of the `filename_repository_t` class and it just serves as a container for the file name and the reference count.

**dec\_ref\_count** ()

Decrease the reference count by 1 and return the new count.

**inc\_ref\_count** ()

Increase the reference count by 1.

**class** `filename_repository_t` (*sha1\_sigs*)

Bases: `object`

File name repository.

This class stores file names and can check whether a file has been modified or not since a previous call. A file name is stored by calling `acquire_filename()` which returns an ID and a signature of the file. The signature can later be used to check if the file was modified by calling `is_file_modified()`. If the file name is no longer required `release_filename()` should be called so that the entry can be removed from the repository.

**acquire\_filename** (*name*)

Acquire a file name and return its id and its signature.

**is\_file\_modified** (*id\_*, *signature*)

Check if the file referred to by *id\_* has been modified.

**release\_filename** (*id\_*)

Release a file name.

**update\_id\_counter** ()

Update the *id\_* counter so that it doesn't grow forever.

**class index\_entry\_t** (*filesigs*, *configsig*)

Bases: `object`

Entry of the index table in the directory cache index.

Each cached header file (i.e. each .cache file) has a corresponding `index_entry_t` object. This object is used to determine whether the cache file with the declarations is still valid or not.

This class is a helper class for the `directory_cache_t` class.

## pygccxml.parser.etree\_scanner module

**class ietree\_scanner\_t** (*xml\_file*, *decl\_factory*, \**args*)

Bases: `pygccxml.parser.scanner.scanner_t`

**access** ()

**calldefs** ()

**characters** (*content*)

Receive notification of character data.

The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity so that the Locator provides useful information.

**declarations** ()

**endDocument** ()

**endElement** (*name*)

**endElementNS** (*name*, *qname*)

Signals the end of an element in namespace mode.

The name parameter contains the name of the element type, just as with the `startElementNS` event.

**endPrefixMapping** (*prefix*)

End the scope of a prefix-URI mapping.

See `startPrefixMapping` for details. This event will always occur after the corresponding `endElement` event, but the order of `endPrefixMapping` events is not otherwise guaranteed.

**enums** ()

**files** ()

**ignorableWhitespace** (*whitespace*)

Receive notification of ignorable whitespace in element content.

Validating Parsers must use this method to report each chunk of ignorable whitespace (see the W3C XML 1.0 recommendation, section 2.10): non-validating parsers may also use this method if they are capable of parsing and using content models.

SAX parsers may return all contiguous whitespace in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

**members** ()

**processingInstruction** (*target, data*)

Receive notification of a processing instruction.

The Parser will invoke this method once for each processing instruction found: note that processing instructions may occur before or after the main document element.

A SAX parser should never report an XML declaration (XML 1.0, section 2.8) or a text declaration (XML 1.0, section 4.3.1) using this method.

**read** ()

**setDocumentLocator** (*locator*)

Called by the parser to give the application a locator for locating the origin of document events.

SAX parsers are strongly encouraged (though not absolutely required) to supply a locator: if it does so, it must supply the locator to the application by invoking this method before invoking any of the other methods in the DocumentHandler interface.

The locator allows the application to determine the end position of any document-related event, even if the parser is not reporting an error. Typically, the application will use this information for reporting its own errors (such as character content that does not match an application's business rules). The information returned by the locator is probably not sufficient for use with a search engine.

Note that the locator will return correct information only during the invocation of the events in this interface. The application should not attempt to use it at any other time.

**skippedEntity** (*name*)

Receive notification of a skipped entity.

The Parser will invoke this method once for each entity skipped. Non-validating processors may skip entities if they have not seen the declarations (because, for example, the entity was declared in an external DTD subset). All processors may skip external entities, depending on the values of the <http://xml.org/sax/features/external-general-entities> and the <http://xml.org/sax/features/external-parameter-entities> properties.

**startDocument** ()

Receive notification of the beginning of a document.

The SAX parser will invoke this method only once, before any other methods in this interface or in DTD-Handler (except for setDocumentLocator).

**startElement** (*name, attrs*)

**startElementNS** (*name, qname, attrs*)

Signals the start of an element in namespace mode.

The name parameter contains the name of the element type as a (uri, localname) tuple, the qname parameter the raw XML 1.0 name used in the source document, and the attrs parameter holds an instance of the Attributes class containing the attributes of the element.

The uri part of the name tuple is None for elements which have no namespace.

**startPrefixMapping** (*prefix, uri*)

Begin the scope of a prefix-URI Namespace mapping.

The information from this event is not necessary for normal Namespace processing: the SAX XML reader will automatically replace prefixes for element and attribute names when the <http://xml.org/sax/features/namespaces> feature is true (the default).

There are cases, however, when applications need to use prefixes in character data or in attribute values, where they cannot safely be expanded automatically; the start/endPrefixMapping event supplies the information to the application to expand prefixes in those contexts itself, if necessary.

Note that start/endPrefixMapping events are not guaranteed to be properly nested relative to each-other: all startPrefixMapping events will occur before the corresponding startElement event, and all endPrefixMapping events will occur after the corresponding endElement event, but their order is not guaranteed.

**types ()**

**xml\_generator\_from\_xml\_file**

Configuration object containing information about the xml generator read from the xml file.

**Returns** configuration object

**Return type** *utils.xml\_generators*

### pygccxml.parser.linker module

**class linker\_t** (*decls, types, access, membership, files, xml\_generator\_from\_xml\_file=None*)

Bases: *pygccxml.declarations.decl\_visitor.decl\_visitor\_t*, *pygccxml.declarations.type\_visitor.type\_visitor\_t*, object

**instance**

**visit\_array ()**

**visit\_bool ()**

**visit\_casting\_operator ()**

**visit\_char ()**

**visit\_class ()**

**visit\_class\_declaration ()**

**visit\_complex\_double ()**

**visit\_complex\_float ()**

**visit\_complex\_long\_double ()**

**visit\_const ()**

**visit\_constructor ()**

**visit\_declarated ()**

**visit\_destructor ()**

**visit\_double ()**

**visit\_elaborated ()**

**visit\_ellipsis ()**

**visit\_enumeration ()**

**visit\_float ()**

**visit\_free\_function ()**

```
visit_free_function_type()
visit_free_operator()
visit_int()
visit_int128()
visit_jboolean()
visit_jbyte()
visit_jchar()
visit_jdouble()
visit_jfloat()
visit_jint()
visit_jlong()
visit_jshort()
visit_long_double()
visit_long_int()
visit_long_long_int()
visit_long_long_unsigned_int()
visit_long_unsigned_int()
visit_member_function()
visit_member_function_type()
visit_member_operator()
visit_member_variable_type()
visit_namespace()
visit_pointer()
visit_reference()
visit_restrict()
visit_short_int()
visit_short_unsigned_int()
visit_signed_char()
visit_typedef()
visit_uint128()
visit_unsigned_char()
visit_unsigned_int()
visit_variable()
visit_void()
visit_volatile()
visit_wchar()
```

## pygccxml.parser.patcher module

**class** `casting_operator_patcher_t`

Bases: `object`

**class** `default_argument_patcher_t` (*enums, cxx\_std*)

Bases: `object`

**fix\_calldef\_decls** (*decls, enums, cxx\_std*)

some times gccxml report typedefs defined in no namespace it happens for example in next situation template< typename X> void ddd(){ typedef typename X::Y YY;} if I will fail on this bug next time, the right way to fix it may be different

**update\_unnamed\_class** (*decls*)

Adds name to class\_t declarations.

If CastXML is being used, the type definitions with an unnamed class/struct are split across two nodes in the XML tree. For example,

```
typedef struct { } cls;
```

produces

```
<Struct id="_7" name="" context="_1" .../> <Typedef id="_8" name="cls" type="_7" context="_1" .../>
```

For each typedef, we look at which class it refers to, and update the name accordingly. This helps the matcher classes finding these declarations. This was the behaviour with gccxml too, so this is important for backward compatibility.

If the castxml epic version 1 is used, there is even an elaborated type declaration between the typedef and the struct/class, that also needs to be taken care of.

**Parameters** `decls` (*list [declaration\_t]*) – a list of declarations to be patched.

**Returns** `None`

## pygccxml.parser.project\_reader module

**class** `COMPILATION_MODE`

Bases: `object`

**ALL\_AT\_ONCE** = 'all at once'

**FILE\_BY\_FILE** = 'file by file'

**create\_cached\_source\_fc** (*header, cached\_source\_file*)

Creates `parser.file_configuration_t` instance, configured to contain path to GCC-XML generated XML file and C++ source file. If XML file does not exists, it will be created and used for parsing. If XML file exists, it will be used for parsing.

**Parameters**

- **header** (*str*) – path to C++ source file
- **cached\_source\_file** (*str*) – path to GCC-XML generated XML file

**Return type** `parser.file_configuration_t`

**create\_gccxml\_fc** (*xml\_file*)

Creates `parser.file_configuration_t` instance, configured to contain path to GCC-XML generated XML file.



**Parameters** `xml_file` (*str*) – path to GCC-XML generated XML file

**Return type** `parser.file_configuration_t`

**create\_source\_fc** (*header*)

Creates `parser.file_configuration_t` instance, configured to contain path to C++ source file

**Parameters** `header` (*str*) – path to C++ source file

**Return type** `parser.file_configuration_t`

**create\_text\_fc** (*text*)

Creates `parser.file_configuration_t` instance, configured to contain Python string, that contains valid C++ code

**Parameters** `text` (*str*) – C++ code

**Return type** `parser.file_configuration_t`

**class file\_configuration\_t** (*data*, *start\_with\_declarations=None*, *content\_type='standard source file'*, *cached\_source\_file=None*)

Bases: `object`

source code location configuration.

The class instance uses “variant” interface to represent the following data:

1. path to a C++ source file
2. path to GCC-XML generated XML file
3. path to a C++ source file and path to GCC-XML generated file

In this case, if XML file does not exists, it will be created. Next time you will ask to parse the source file, the XML file will be used instead.

Small tip: you can setup your makefile to delete XML files every time, the relevant source file was changed.

4. Python string, that contains valid C++ code

There are few functions, that will help you to construct `file_configuration_t` object:

- `create_source_fc()`
- `create_gccxml_fc()`
- `create_cached_source_fc()`
- `create_text_fc()`

**class CONTENT\_TYPE**

Bases: `object`

**CACHED\_SOURCE\_FILE** = ‘cached source file’

**GCCXML\_GENERATED\_FILE** = ‘gccxml generated file’

**STANDARD\_SOURCE\_FILE** = ‘standard source file’

**TEXT** = ‘text’

`file_configuration_t.cached_source_file`

`file_configuration_t.content_type`

`file_configuration_t.data`

`file_configuration_t.start_with_declarations`

**class** `project_reader_t` (*config*, *cache=None*, *decl\_factory=None*)  
Bases: `object`

parses header files and returns the contained declarations

**static** `get_os_file_names` (*files*)  
returns file names

**Parameters** `files` (*list*) – list of strings and/or `file_configuration_t` instances.

**read\_files** (*files*, *compilation\_mode='file by file'*)  
parses a set of files

**Parameters**

- **files** (*list*) – list of strings and/or `file_configuration_t` instances.
- **compilation\_mode** (*COMPILATION\_MODE*) – determines whether the files are parsed individually or as one single chunk

**Return type** [`declaration_t`]

**read\_string** (*content*)  
Parse a string containing C/C++ source code.

**Parameters** `content` (*str*) – C/C++ source code.

**Return type** `Declarations`

**read\_xml** (*file\_configuration*)  
parses C++ code, defined on the file\_configurations and returns GCCXML generated file content

**xml\_generator\_from\_xml\_file**  
Configuration object containing information about the xml generator read from the xml file.

**Returns** configuration object

**Return type** `utils.xml_generators`

## pygccxml.parser.scanner module

**class** `scanner_t` (*xml\_file*, *decl\_factory*, *config*, *\*args*)  
Bases: `xml.sax.handler.ContentHandler`

**access** ()

**calldefs** ()

**characters** (*content*)  
Receive notification of character data.

The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity so that the Locator provides useful information.

**declarations** ()

**endDocument** ()

**endElement** (*name*)

**endElementNS** (*name, qname*)

Signals the end of an element in namespace mode.

The name parameter contains the name of the element type, just as with the startElementNS event.

**endPrefixMapping** (*prefix*)

End the scope of a prefix-URI mapping.

See startPrefixMapping for details. This event will always occur after the corresponding endElement event, but the order of endPrefixMapping events is not otherwise guaranteed.

**enums** ()**files** ()**ignorableWhitespace** (*whitespace*)

Receive notification of ignorable whitespace in element content.

Validating Parsers must use this method to report each chunk of ignorable whitespace (see the W3C XML 1.0 recommendation, section 2.10); non-validating parsers may also use this method if they are capable of parsing and using content models.

SAX parsers may return all contiguous whitespace in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

**members** ()**processingInstruction** (*target, data*)

Receive notification of a processing instruction.

The Parser will invoke this method once for each processing instruction found: note that processing instructions may occur before or after the main document element.

A SAX parser should never report an XML declaration (XML 1.0, section 2.8) or a text declaration (XML 1.0, section 4.3.1) using this method.

**read** ()**setDocumentLocator** (*locator*)

Called by the parser to give the application a locator for locating the origin of document events.

SAX parsers are strongly encouraged (though not absolutely required) to supply a locator: if it does so, it must supply the locator to the application by invoking this method before invoking any of the other methods in the DocumentHandler interface.

The locator allows the application to determine the end position of any document-related event, even if the parser is not reporting an error. Typically, the application will use this information for reporting its own errors (such as character content that does not match an application's business rules). The information returned by the locator is probably not sufficient for use with a search engine.

Note that the locator will return correct information only during the invocation of the events in this interface. The application should not attempt to use it at any other time.

**skippedEntity** (*name*)

Receive notification of a skipped entity.

The Parser will invoke this method once for each entity skipped. Non-validating processors may skip entities if they have not seen the declarations (because, for example, the entity was declared in an external DTD subset). All processors may skip external entities, depending on the values of the <http://xml.org/sax/features/external-general-entities> and the <http://xml.org/sax/features/external-parameter-entities> properties.

**startDocument ()**

Receive notification of the beginning of a document.

The SAX parser will invoke this method only once, before any other methods in this interface or in DTD-Handler (except for setDocumentLocator).

**startElement (name, attrs)****startElementNS (name, qname, attrs)**

Signals the start of an element in namespace mode.

The name parameter contains the name of the element type as a (uri, localname) tuple, the qname parameter the raw XML 1.0 name used in the source document, and the attrs parameter holds an instance of the Attributes class containing the attributes of the element.

The uri part of the name tuple is None for elements which have no namespace.

**startPrefixMapping (prefix, uri)**

Begin the scope of a prefix-URI Namespace mapping.

The information from this event is not necessary for normal Namespace processing: the SAX XML reader will automatically replace prefixes for element and attribute names when the <http://xml.org/sax/features/namespaces> feature is true (the default).

There are cases, however, when applications need to use prefixes in character data or in attribute values, where they cannot safely be expanded automatically; the start/endPrefixMapping event supplies the information to the application to expand prefixes in those contexts itself, if necessary.

Note that start/endPrefixMapping events are not guaranteed to be properly nested relative to each-other: all startPrefixMapping events will occur before the corresponding startElement event, and all endPrefixMapping events will occur after the corresponding endElement event, but their order is not guaranteed.

**types ()****xml\_generator\_from\_xml\_file**

Configuration object containing information about the xml generator read from the xml file.

**Returns** configuration object

**Return type** *utils.xml\_generators*

## pygccxml.parser.source\_reader module

**bind\_aliases (decls)**

This function binds between class and it's typedefs.

Deprecated since 1.9.0, will be removed in 2.0.0

**Parameters** **decls** – list of all declarations

**Return type** None

**class source\_reader\_t (configuration, cache=None, decl\_factory=None)**

Bases: object

This class reads C++ source code and returns the declarations tree.

This class is the only class that works directly with GCC-XML or CastXML.

It has only one responsibility: it calls GCC-XML with a source file specified by the user and creates declarations tree. The implementation of this class is split to two classes:

1. **`scanner_t`** - this class scans the “XML” file, generated by GCC-XML or CastXML and creates `pygccxml` declarations and types classes. After the XML file has been processed declarations and type class instances keeps references to each other using GCC-XML or CastXML generated id’s.

2. **`linker_t`** - this class contains logic for replacing GCC-XML or CastXML generated ids with references to declarations or type class instances.

**`create_xml_file`** (*source\_file*, *destination=None*)

This method will generate a xml file using an external tool.

The external tool can be either gccxml or castxml. The method will return the file path of the generated xml file.

#### Parameters

- **`source_file`** (*str*) – path to the source file that should be parsed.
- **`destination`** (*str*) – if given, will be used as target file path for GCC-XML or CastXML.

**Return type** path to xml file.

**`create_xml_file_from_string`** (*content*, *destination=None*)

Creates XML file from text.

#### Parameters

- **`content`** (*str*) – C++ source code
- **`destination`** (*str*) – file name for GCC-XML generated file

**Return type** returns file name of GCC-XML generated file

**`join_declarations`** (*namespace*)

**`read_cpp_source_file`** (*source\_file*)

Reads C++ source file and returns declarations tree

**Parameters** **`source_file`** (*str*) – path to C++ source file

**`read_file`** (*source\_file*)

**`read_string`** (*content*)

Reads a Python string that contains C++ code, and return the declarations tree.

**`read_xml_file`** (*xml\_file*)

Read generated XML file.

**Parameters** **`xml_file`** (*str*) – path to xml file

**Return type** declarations tree

**`xml_generator_from_xml_file`**

Configuration object containing information about the xml generator read from the xml file.

**Returns** configuration object

**Return type** `utils.xml_generators`

## pygccxml.utils package

The utils package contains tools used internally by pygccxml.

## Submodules

### pygccxml.utils.utils module

Logger classes and a few convenience methods.

**class DeprecationWrapper** (*new\_target, old\_name, new\_name, version*)  
Bases: object

A small wrapper class useful when deprecation classes.

This class is not part of the public API.

**class cached** (*method*)  
Bases: property

Convert a method into a cached attribute.

**deleter** ()  
Descriptor to change the deleter on a property.

**fdel**

**fget**

**fset**

**getter** ()  
Descriptor to change the getter on a property.

**reset** ()

**setter** ()  
Descriptor to change the setter on a property.

**contains\_parent\_dir** (*fpath, dirs*)  
Returns true if paths in dirs start with fpath.

Precondition: dirs and fpath should be normalized before calling this function.

**create\_temp\_file\_name** (*suffix, prefix=None, dir=None, directory=None*)  
Small convenience function that creates temporary files.

This function is a wrapper around the Python built-in function `tempfile.mkstemp`.

**class cxx\_standard** (*cflags*)  
Bases: object

Helper class for parsing the C++ standard version.

This class holds the C++ standard version the XML generator has been configured with, and provides helpers functions for querying C++ standard version related information.

**is\_cxx03**  
Returns true if `-std=c++03` is being used

**is\_cxx11**  
Returns true if `-std=c++11` is being used

**is\_cxx11\_or\_greater**  
Returns true if `-std=c++11` or a newer standard is being used

**is\_cxx14**  
Returns true if `-std=c++14` is being used

**is\_cxx14\_or\_greater**

Returns true if -std=c++14 or a newer standard is being used

**is\_cxx1z**

Returns true if -std=c++1z is being used

**is\_implicit**

Indicates whether a -std=c++xx was specified

**stdcxx**

Returns the -std=c++xx option passed to the constructor

**find\_xml\_generator** (*name=None*)

Try to find a c++ parser. Returns path and name.

**Parameters** *name* (*str*) – name of the c++ parser: castxml or gccxml

If no name is given the function first looks for castxml, then for gccxml. If no c++ parser is found the function raises an exception.

**get\_architecture** ()

Returns computer architecture: 32 or 64.

The guess is based on maxint.

**get\_tr1** (*name*)

In libstd++ the tr1 namespace needs special care.

Return either an empty string or tr1::, useful for appending to search patterns.

**Parameters** *name* (*str*) – the name of the declaration

**Returns** an empty string or “tr1::”

**Return type** *str*

**is\_str** (*string*)

Python 2 and 3 compatible string checker.

**Parameters** *string* (*str* | *basestring*) – the string to check

**Returns** True or False

**Return type** *bool*

**class loggers**

Bases: *object*

Class-namespace, defines a few loggers classes, used in the project.

**all\_loggers** = [*<logging.Logger object>*, *<logging.Logger object>*, *<logging.Logger object>*, *<logging.Logger object>*, ...]

Contains all logger classes, defined by the class.

**cxx\_parser** = *<logging.Logger object>*

Logger for C++ parser functionality

If you set this logger level to DEBUG, you will be able to see the exact command line, used to invoke GCC-XML and errors that occurs during XML parsing

**declarations\_cache** = *<logging.Logger object>*

Logger for declarations tree cache functionality

If you set this logger level to DEBUG, you will be able to see what is exactly happens, when you read the declarations from cache file. You will be able to decide, whether it worse for you to use this or that cache strategy.

**pdb\_reader** = <logging.Logger object>

Logger for MS .pdb file reader functionality

**queries\_engine** = <logging.Logger object>

Logger for query engine functionality.

If you set this logger level to DEBUG, you will be able to see what queries you do against declarations tree, measure performance and may be even to improve it. Query engine reports queries and whether they are optimized or not.

**root** = <logging.Logger object>

Root logger exists for your convenience only.

**static set\_level** (*level*)

Set the same logging level for all the loggers at once.

**normalize\_path** (*some\_path*)

Return os.path.normpath(os.path.normcase(some\_path)).

**remove\_file\_no\_raise** (*file\_name*, *config*)

Removes file from disk if exception is raised.

## pygccxml.utils.xml\_generators module

**class xml\_generators** (*logger*, *gccxml\_cvs\_revision=None*, *castxml\_format=None*)

Bases: object

Helper class for the xml generator versions

**get\_string\_repr** ()

Get a string identifier for the current type of xml generator

**Returns** identifier

**Return type** str

**is\_castxml**

Is the current xml generator castxml?

**Returns** is castxml being used?

**Return type** bool

**is\_castxml1**

Is the current xml generator castxml (with output format version 1)?

**Returns** is castxml (with output format version 1) being used?

**Return type** bool

**is\_gccxml**

Is the current xml generator gccxml?

**Returns** is gccxml being used?

**Return type** bool

**is\_gccxml\_06**

Is the current xml generator gccxml (version 0.6)?

**Returns** is gccxml 0.6 being used?

**Return type** bool



**is\_gccxml\_07**

Is the current xml generator gccxml (version 0.7)?

**Returns** is gccxml 0.7 being used?

**Return type** bool

**is\_gccxml\_09**

Is the current xml generator gccxml (version 0.9)?

**Returns** is gccxml 0.9 being used?

**Return type** bool

**is\_gccxml\_09\_buggy**

Is the current xml generator gccxml (version 0.9 - buggy)?

**Returns** is gccxml 0.9 (buggy) being used?

**Return type** bool

**xml\_output\_version**

The current xml output version for the parsed file.

**Returns** the xml output version

**Return type** str

## Building the documentation

### Building the documentation locally

You can build the documentation yourself. In order for this to work you need sphinx doc (<http://sphinx-doc.org>) and the readthedocs theme:

```
pip install sphinx
```

```
pip install sphinx_rtd_theme
```

Then just run the following command in the root folder:

```
make html
```

This will build the documentation locally in the `docs/_build/html` folder.

For each commit on the master and develop branches, the documentation is automatically built and can be found here: <https://readthedocs.org/projects/pygccxml/>

## Declarations query API

### Introduction

You parsed the source files. Now you have to do some real work with the extracted information, right? pygccxml provides very powerful and simple interface to query about extracted declarations.

Just an example. I want to select all member functions, which have 2 arguments. I don't care about first argument type, but I do want second argument type to be a reference to an integer. More over, I want those functions names to end with "impl" string and they should be protected or private.

```
#global_ns is the reference to an instance of namespace_t object, that
#represents global namespace
query = declarations.custom_matcher_t( lambda mem_fun: mem_fun.name.endswith( 'impl' )
query = query & ~declarations.access_type_matcher_t( 'public' )
global_ns.member_functions( function=query, arg_types=[None, 'int &'] )
```

The example is complex, but still readable. In many cases you will find yourself, looking for one or many declarations, using one or two declaration properties. For example:

```
global_ns.namespaces( 'details' )
```

This call will return all namespaces with name ‘details’.

## User interface

As you already know, `pygccxml.declarations` package defines the following classes:

- `scopeddef_t` - base class for all classes, that can contain other declarations
- `namespace_t` - derives from `scopeddef_t` class, represents C++ namespace
- `class_t` - derives from `scopeddef_t` class, represents C++ class/struct/union.

So, the query methods defined on `scopeddef_t` class could be used on instances of `class_t` and `namespace_t` classes. I am sure you knew that.

## Usage examples

I will explain the usage of `member_function` and `member_functions` methods. The usage of other methods is very similar to them. Here is definition of those methods:

```
def member_function( self,
                     name=None,
                     function=None,
                     return_type=None,
                     arg_types=None,
                     header_dir=None,
                     header_file=None,
                     recursive=None )

mem_fun = member_function #just an alias

def member_functions( self,
                     name=None,
                     function=None,
                     return_type=None,
                     arg_types=None,
                     header_dir=None,
                     header_file=None,
                     recursive=None,
                     allow_empty=None )
mem_funs = member_functions
```

As you can see, from the method arguments you can search for member function by:

- name

Python string, that contains member function name or full name.

```
do_smth = my_class.member_function( 'do_smth' )
do_smth = my_class.member_function( 'my_namespace::my_class::do_smth' )
```

- `function`

Python callable object. You would use this functionality, if you need to build custom query. This object will be called with one argument - declaration, and it should return True or False.

```
impls = my_class.member_functions( lambda decl: decl.name.endswith( 'impl' ) )
```

`impls` will contain all member functions, that their name ends with “impl”.

- `return_type`

the function return type. This argument can be string or an object that describes C++ type.

```
mem_funcs = my_class.member_functions( return_type='int' )

i = declarations.int_t()
ref_i = declarations.reference_t( i )
const_ref_i = declarations.const_t( ref_i )
mem_funcs = my_class.member_functions( return_type=const_ref_int )
```

- `arg_types`

Python list that contains description of member function argument types. This list could be a mix of Python strings and objects that describes C++ type. Size of list says how many arguments function should have. If you want to skip some argument type from within comparison, you put None, into relevant position within the list.

```
mem_funcs = my_class.member_functions( arg_types=[ None, 'int' ] )
```

`mem_funcs` will contain all member functions, which have two arguments and type of second argument is `int`.

- `header_dir`

Python string, that contains full path to directory, which contains file, which contains the function declaration

```
mem_funcs = my_namespace.member_functions( header_dir='/home/roman/xyz' )
```

- `header_file`

Python string, that contains full path to file, which contains the function declaration.

```
mem_funcs = my_namespace.member_functions( header_dir='/home/roman/xyz/xyz.hpp' )
```

- `recursive`

Python boolean object.

If `recursive` is True, then member function will be also searched within internal declarations.

If `recursive` is False, then member function will be searched only within current scope.

What happen if `recursive` is None? Well. `scopedef_t` class defines `RECURSIVE_DEFAULT` variable. Its initial value is True. So, if you don't pass `recursive` argument, the value of `RECURSIVE_DEFAULT` variable will be used. This “yet another level of indirection” allows you to configure pygccxml “select” functions in one place for all project.

- `allow_empty`

Python boolean object, it says pygccxml what to do if query returns empty.

If `allow_empty` is `False`, then exception `RuntimeError( "Multi declaration query returned 0 declarations." )` will be raised

`allow_empty` uses same technique as `recursive`, to allow you to customize the behavior project-wise. The relevant class variable name is `ALLOW_EMPTY_MDECL_WRAPPER`. Its initial value is `False`.

Now, when you understand, how to call those functions, I will explain what they return.

`member_function` will always return reference to desired declaration. If declaration could not be found or there are more then one declaration that match query `RuntimeError` exception will be raised.

Return value of `member_functions` is not Python list or set, but instance of `mdecl_wrapper_t` class. This class allows you to work on all selected objects at once. I will give an example from another project - <https://pypi.python.org/pypi/pyplusplus/>. In order to help `Boost.Python` to manage objects life time, all functions should have call policies. For example:

```
struct A{
    A* clone() const { return new A(); }
    ...
};
```

```
struct B{
    B* clone() const { return new B(); }
    ...
};
```

`clone` member function call policies is `return_value_policy<manage_new_object>()`. The following code applies the call policies on all `clone` member functions within the project:

```
#global_ns - instance of namespace_t class, that contains reference to global_
↳namespace
clone = global_ns.member_functions( 'clone' )
clone.call_policies = return_value_policy( manage_new_object )
```

Another example, from <https://pypi.python.org/pypi/pyplusplus/> project. Sometimes it is desirable to exclude declaration, from being exported to Python. The following code will exclude `clone` member function from being exported:

```
global_ns.member_functions( 'clone' ).exclude()
```

As you can see this class allows you to write less code. Basically using this class you don't have to write loops. It will do it for you. Also if you insist to write loops, `mdecl_wrapper_t` class implements `__len__`, `__getitem__` and `__iter__` methods. So you can write the following code:

```
for clone in global_ns.member_functions( 'clone' ):
    print clone.parent.name
```

## Implementation details

### Performance

For big projects, performance is critical. When you finished to build/change declarations tree, then you can call `scopedef_t.init_optimizer` method. This method will initialize few data structures, that will help to minimize the number of compared declarations. The price you are going to pay is memory usage.

## Data structures

Here is a short explanation of what data structures is initialized.

- `scopedef_t._type2decls`, `scopedef_t._type2decls_nr`

Python dictionary, that contains mapping between declaration type and declarations in the current scope.

`scopedef_t.type2decls_nr` contains only declaration from the current scope. `scopedef_t.type2decls` contains declarations from the current scope and its children

- `scopedef_t._type2name2decls`, `scopedef_t._type2name2decls_nr`

Python dictionary, that contains mapping between declaration type and another dictionary. This second dictionary contains mapping between a declaration name and declaration.

`scopedef_t.type2name2decls_nr` contains only declaration from the current scope. `scopedef_t.type2name2decls` contains declarations from the current scope and its children

- `scopedef_t._all_decls`

A flat list of all declarations, including declarations from the children scopes.

Except `scopedef_t.decl` and `scopedef_t.decls` methods, all other queries have information about declaration type.

If you include `name` into your query, you will get the best performance.

## More information

I think, I gave you the important information. If you need definition of some query method, you can take a look on API documentation or into source code.

## Design overview

pygccxml has 4 packages:

- *declarations*

This package defines classes that describe C++ declarations and types.

- *parser*

This package defines classes that parse [GCC-XML](#) or [CastXML](#) generated files. It also defines a few classes that will help you unnecessary parsing of C++ source files.

- *utils*

This package defines a few functions useful for the whole project, but which are mainly used internally by pygccxml.

### declarations package

Please take a look on the UML diagram. This UML diagram describes almost all classes defined in the package and their relationship. `declarations` package defines two hierarchies of class:

1. types hierarchy - used to represent a C++ type
2. declarations hierarchy - used to represent a C++ declaration

## Types hierarchy

Types hierarchy is used to represent an arbitrary type in C++. class `type_t` is the base class.

### `type_traits`

Are you aware of `boost::type_traits` library? The `boost::type_traits` library contains a set of very specific traits classes, each of which encapsulate a single trait from the C++ type system; for example, is a type a pointer or a reference? Or does a type have a trivial constructor, or a const-qualifier?

pygccxml implements a lot of functionality from the library:

- a lot of algorithms were implemented
  - `is_same`
  - `is_enum`
  - `is_void`
  - `is_const`
  - `is_array`
  - `is_pointer`
  - `is_volatile`
  - `is_integral`
  - `is_reference`
  - `is_arithmetic`
  - `is_convertible`
  - `is_fundamental`
  - `is_floating_point`
  - `is_base_and_derived`
  - `is_unary_operator`
  - `is_binary_operator`
  - `remove_cv`
  - `remove_const`
  - `remove_alias`
  - `remove_pointer`
  - `remove_volatile`
  - `remove_reference`
  - `has_trivial_copy`
  - `has_trivial_constructor`
  - `has_any_non_copyconstructor`

For a full list of implemented algorithms, please consult API documentation.

- a lot of unit tests has been written base on unit tests from the `boost::type_traits` library.

If you are going to build code generator, you will find `type_traits` very handy.

## Declarations hierarchy

A declaration hierarchy is used to represent an arbitrary C++ declaration. Basically, most of the classes defined in this package are just “set of properties”.

`declaration_t` is the base class of the declaration hierarchy. Every declaration has `parent` property. This property keeps a reference to the scope declaration instance, in which this declaration is defined.

The `scopeddef_t` class derives from `declaration_t`. This class is used to say - “I may have other declarations inside”. The “composite” design pattern is used here. `class_t` and `namespace_t` declaration classes derive from the `scopeddef_t` class.

## parser package

Please take a look on parser package UML diagram . Classes defined in this package, implement parsing and linking functionality. There are few kind of classes defined by the package:

- classes, that implements parsing algorithms of [GCC-XML](#) generated XML file
- parser configuration classes
- cache - classes, those one will help you to eliminate unnecessary parsing
- patchers - classes, which fix [GCC-XML](#) generated declarations. ( Yes, sometimes GCC-XML generates wrong description of C++ declaration. )

## Parser classes

`source_reader_t` - the only class that have a detailed knowledge about [GCC-XML](#). It has only one responsibility: it calls [GCC-XML](#) with a source file specified by user and creates declarations tree. The implementation of this class is split to 2 classes:

1. `scanner_t` - this class scans the “XML” file, generated by [GCC-XML](#) and creates pygccxml declarations and types classes. After the xml file has been processed declarations and type class instances keeps references to each other using [GCC-XML](#) generated ids.
2. `linker_t` - this class contains logic for replacing [GCC-XML](#) generated ids with references to declarations or type class instances.

Both those classes are implementation details and should not be used by user. Performance note: `scanner_t` class uses Python `xml . sax` package in order to parse XML. As a result, `scanner_t` class is able to parse even big XML files pretty quick.

`project_reader_t` - think about this class as a linker. In most cases you work with few source files. [GCC-XML](#) does not supports this mode of work. So, pygccxml implements all functionality needed to parse few source files at once. `project_reader_t` implements 2 different algorithms, that solves the problem:

1. `project_reader_t` creates temporal source file, which includes all the source files.
2. `project_reader_t` parse separately every source file, using `source_reader_t` class and then joins the resulting declarations tree into single declarations tree.

Both approaches have different trades-off. The first approach does not allow you to reuse information from already parsed source files. While the second one allows you to setup cache.

## Parser configuration classes

`gccxml_configuration_t` - a class, that accumulates all the settings needed to invoke [GCC-XML](#):

`file_configuration_t` - a class, that contains some data and description how to treat the data.  
`file_configuration_t` can contain reference to the the following types of data:

1. path to C++ source file
2. path to [GCC-XML](#) generated XML file
3. path to C++ source file and path to [GCC-XML](#) generated XML file

In this case, if XML file does not exists, it will be created. Next time you will ask to parse the source file, the XML file will be used instead.

Small tip: you can setup your makefile to delete XML files every time, the relevant source file has changed.

4. Python string, that contains valid C++ code

There are few functions that will help you to construct `file_configuration_t` object:

- `def create_source_fc( header )`  
header contains path to C++ source file
- `def create_gccxml_fc( xml_file )`  
xml\_file contains path to [GCC-XML](#) generated XML file
- `def create_cached_source_fc( header, cached_source_file )`
  - header contains path to C++ source file
  - xml\_file contains path to [GCC-XML](#) generated XML file
- `def create_text_fc( text )`  
text - Python string, that contains valid C++ code

## Cache classes

There are few cache classes, which implements different cache strategies.

1. `file_configuration_t` class, that keeps path to C++ source file and path to [GCC-XML](#) generated XML file.
2. `file_cache_t` class, will save all declarations from all files within single binary file.
3. `directory_cache_t` class will store one index file called “index.dat” which is always read by the cache when the cache object is created. Each header file will have its corresponding \*.cache file that stores the declarations found in the header file. The index file is used to determine whether a \*.cache file is still valid or not (by checking if one of the dependent files (i.e. the header file itself and all included files) have been modified since the last run).

In some cases, `directory_cache_t` class gives much better performance, than `file_cache_t`. Many thanks to Matthias Baas for its implementation.

**Warning:** when pygccxml writes information to files, using cache classes, it does not write any version information. It means, that when you upgrade pygccxml you have to delete all your cache files. Otherwise you will get very strange errors. For example: missing attribute.



## Patchers

Well, GCC-XML has few bugs, which could not be fixed from it. For example

```
namespace ns1{ namespace ns2{
    enum fruit{ apple, orange };
} }
```

```
void fix_enum( ns1::ns2::fruit arg=ns1::ns2::apple );
```

GCC-XML will report the default value of `arg` as `apple`. Obviously this is an error. pygccxml knows how to fix this bug.

This is not the only bug, which could be fixed, there are few of them. pygccxml introduces few classes, which knows how to deal with specific bug. More over, those bugs are fixed, only if I am 101% sure, that this is the right thing to do.

## utils package

Use internally by pygccxml. Some methods/classes may be still usefull: `loggers`, `find_xml_generator`

## Summary

That's all. I hope I was clear, at least I tried. Any way, pygccxml is an open source project. You always can take a look on the source code. If you need more information please read API documentation.

## Who is using pygccxml?

### Users

- The Insight Toolkit is using pygccxml (<http://www.itk.org>).
- `PyBindGen` - is a Python module that is geared to generating C/C++ code that binds a C/C++ library for Python.
- your project name ... :-)

### pygccxml in blogs

- <http://blog.susheel.space.com/?p=88>  
” ... I have used pygccxml for parsing c++ code, it was a lot of fun to use “
- <http://cysquatch.net/blog/2007/09/01/c-code-metrics-with-pygccxml>  
pygccxml is used to calculate the Weighted Methods per Class (WMC) metric.
- <http://www.garagegames.com/blogs/4280/13907>  
pygccxml is used to generate input files for SIP code generator.
- [http://blogs.sun.com/thorsten/entry/more\\_on\\_source\\_code\\_grokking](http://blogs.sun.com/thorsten/entry/more_on_source_code_grokking)  
Short listing of C++ parsers and their description.

## C++ Reflection

### Links

- [CppReflect](#) - extracts reflection information from executables, which were build with debug information. It works with executables that has COFF or ELF format.
- [XTI An Extended Type Information Library](#) - Bjarne Stroustrup talk about adding reflection information to C++ program.

## Releasing

To build a new release, modify the version number in:

```
pygccxml/__init__.py
```

This version number will then automatically be used to build the documentation and by the setup.py script when building the wheels.

Do not forget to document the changes in the `CHANGELOG.md` file.

### Uploading to pypi

The documentation for the building and uploading can be found here: [pypi](#)

The wheels are built with:

```
python setup.py bdist_wheel --universal
```

They are uploaded with:

```
twine upload dist/*
```

## History and Credits

### History

The original author and maintainer for pygccxml was Roman Yakovenko (2004-2011).

Holger Frydrych forked the project to work on python 3 support. Finally, Mark Moll forked the project a second time to carry on the work of porting the code to python 3 (keeping it compatible with python 2).

In Mai 2014, Michka Popoff and the Insight Software Consortium revived pygccxml by setting up a git repository on GitHub, hosted along with gccxml.

The full changelog can be found in the `CHANGELOG.md` file.

### Contributors

Thanks to all the people that have contributed patches, bug reports and suggestions, or supported this project.

- Roman Yakovenko (original author)
- Michka Popoff (actual maintainer)

A special thanks to the Insight Software Consortium for their help and collaboration, especially:

- Brad King
- Matt McCormick

Contributors can be found on github's contributors page: <https://github.com/gccxml/pygccxml/graphs/contributors>

Here is the original list of contributors from before the GitHub migration.

- Roman Yakovenko's wife - Yulia
- Mark Moll
- Holger Frydrych
- John Pallister
- Matthias Baas
- Allen Bierbaum
- Georgiy Dernovoy
- Darren Garnier
- Gottfried Ganssauge
- Gaetan Lehmann
- Martin Preisler
- Miguel Lobo
- Jeremy Sanders
- Ben Schleimer
- Gustavo Carneiro
- Christopher Bruns
- Alejandro Dubrovsky
- Aron Xu

## **GCC-XML 0.7 → 0.9 upgrade issues (Legacy)**

### **Introduction**

This page is kept here for historical reasons. CastXML is the recommended xml generator and GCC-XML is being phased out. This page will be removed in version 2.0.0 of pygccxml.

Recently, GCC-XML internal parser was updated to GCC 4.2. The internal representation of source code, provided by GCC's parser, has changed a lot and few backward compatibility issues were introduced. In this document, I will try to cover all problems you may encounter.

### **Default constructor**

GCC-XML 0.9 doesn't report compiler generated default and copy constructors as an implicit ones.

If you rely heavily on their existence in the generated XML, I suggest you to switch to `has_trivial_constructor` and to `has_trivial_copy` functions.

## Pointer to member variable

Previous version of GCC-XML reported pointer to member variable as a “PointerType” with reference to “OffsetType”. The new version removes “PointerType” from this sequence.

C++ code:

```
struct xyz_t{
    int do_smth( double );
    int m_some_member;
};

typedef int (xyz_t::*mfun_ptr_t)( double );

typedef int (xyz_t::*mvar_ptr_t);
```

GCC-XML 0.7:

```
<Typedef id="_6" name="mfun_ptr_t" type="_5" />
<PointerType id="_5" type="_128" size="32" align="32"/>
<MethodType id="_128" basetype="_7" returns="_136">
    <Argument type="_140"/>
</MethodType>

<Typedef id="_4" name="mvar_ptr_t" type="_3" />
<PointerType id="_3" type="_127" size="32" align="32"/>
<OffsetType id="_127" basetype="_7" type="_136" size="32" align="32"/>
```

GCC-XML 0.9:

```
<Typedef id="_97" name="mfun_ptr_t" type="_96" />
<PointerType id="_96" type="_147" size="32" align="32"/>
<MethodType id="_147" basetype="_92" returns="_131">
    <Argument type="_127"/>
</MethodType>

<Typedef id="_52" name="mvar_ptr_t" type="_139" />
<OffsetType id="_139" basetype="_92" type="_131" size="32" align="32"/>
```

pygccxml handles this issue automatically, you don’t have to change your code.

## Constant variable value

GCC-XML 0.9 uses suffix to report the constant variable value

For example:

```
const long unsigned int initialized = 10122004;
```

GCC-XML 0.9 will report the `initialized` value as `10122004ul`, while GCC-XML 0.7 as `10122004`.

pygccxml handles this problem automatically, you don’t have to change your code.

## Free and member function default arguments

Both versions of GCC-XML have a few issues, related to default arguments. GCC-XML 0.9 fixes some issues, but introduces another ones. Take a look on the following examples:

- Example 1

```
void fix_numeric( ull arg=(ull)-1 );
```

GCC-XML 0.7

```
<Argument name="arg" type="_7" default="0xffffffffffffffff"/>
```

GCC-XML 0.9

```
<Argument name="arg" type="_103" default="0xffffffffffffffffu"/>
```

- Example 2

```
void fix_function_call( int i=calc( 1,2,3) );
```

GCC-XML 0.7

```
<Argument name="i" type="_9" default="function_call::calc(int, int, int)(1, 2, 3)
↪"/>
```

GCC-XML 0.9

```
<Argument name="i" type="_34" default="function_call::calc(1, 2, 3)"/>
```

- Example 3

```
void typedef__func( const typedef_::alias& position = typedef_::alias() );
```

GCC-XML 0.7

```
<Argument name="position" type="_1458" default="alias()"/>
```

GCC-XML 0.9

```
<Argument name="position" type="_1703" default="typedef_::original_name()"/>
```

- Example 4

```
void typedef__func2( const typedef_::alias& position = alias() );
```

GCC-XML 0.7

```
<Argument name="position" type="_1458" default="alias()"/>
```

GCC-XML 0.9

```
<Argument name="position" type="_1703" default="typedef_::original_name()"/>
```

- Example 5

```
node* clone_tree( const std::vector<std::string> &types=std::vector<std::string>
↪() );
```

GCC-XML 0.7

```
<Argument name="types" type="_3336" default="vector<std::basic_string<char,
↪std::char_traits<char>, std::allocator<char> &gt;, std::allocator<
↪std::basic_string<char, std::char_traits<char>, std::allocator<char&
↪gt; &gt; &gt; &gt;((&allocator<std::basic_string<char, std::char_
↪traits<char>, std::allocator<char> &gt; &gt; &gt;()))"/>
```

## GCC-XML 0.9

```
<Argument name="types" type="_3096" default="std::vector<std::basic_string<
↪char, std::char_traits<char>;, std::allocator<char> >>;,
↪std::allocator<std::basic_string<char, std::char_traits<char>;,
↪std::allocator<char> >> >> >>(((const std::allocator<std::basic_
↪string<char, std::char_traits<char>;, std::allocator<char> >> &
↪gt;& &)((const std::allocator<std::basic_string<char, std::char_traits<
↪lt;char>;, std::allocator<char> >> >> >>*)(& std::allocator<
↪std::basic_string<char, std::char_traits<char>;, std::allocator<char&
↪gt; >> >> >>)))/>
```

Basically pygccxml can't help you here. The good news is that you always can change the default value expression from the script:

```
#f is "calldef_t" instance
for arg in f.arguments:
    arg.default_value = <<<new default value or None>>>
```

## Name mangling

GCC-XML 0.9 mangles names different than the previous one. This change is the most dramatic one, because it may require from you to change the code.

Consider the following C++ code:

```
template< unsigned long i1>
struct item_t{
    static const unsigned long v1 = i1;
};

struct buggy{
    typedef unsigned long ulong;
    typedef item_t< ulong( 0xDEECE66DUL ) | (ulong(0x5) << 32) > my_item_t;
    my_item_t my_item_var;
};
```

| generated data       | GCC-XML 0.7            | GCC-XML 0.9            |
|----------------------|------------------------|------------------------|
| class name           | item_t<0xdeece66d>     | item_t<-554899859ul>   |
| class mangled name   | 6item_tILm3740067437EE | 6item_tILm3740067437EE |
| class demangled name | item_t<3740067437l>    | item_t<3740067437ul>   |

pygccxml uses class demangled name as a “name” of the class. This was done to overcome few bugs GCC-XML has, when it works on libraries with extreme usage of templates.

As you can see the name of the class is different. pygccxml is unable to help you in such situations. I suggest you to use query API strict mode. This is the default one. If the class/declaration with the given name could not be found, it will raise an error with clear description of the problem.

You can also to print the declarations tree to `stdout` and find out the name of the class/declaration from it.

## CHAPTER 7

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





### p

pygccxml, 47  
pygccxml.declarations, 47  
pygccxml.declarations.algorithm, 48  
pygccxml.declarations.algorithms\_cache, 49  
pygccxml.declarations.byte\_info, 50  
pygccxml.declarations.call\_invocation, 50  
pygccxml.declarations.calldef, 51  
pygccxml.declarations.calldef\_members, 53  
pygccxml.declarations.calldef\_types, 67  
pygccxml.declarations.class\_declaration, 68  
pygccxml.declarations.container\_traits, 75  
pygccxml.declarations.cpptypes, 77  
pygccxml.declarations.decl\_factory, 100  
pygccxml.declarations.decl\_printer, 101  
pygccxml.declarations.decl\_visitor, 102  
pygccxml.declarations.declaration, 102  
pygccxml.declarations.declaration\_utils, 104  
pygccxml.declarations.declarations\_matchers, 104  
pygccxml.declarations.elaborated\_info, 106  
pygccxml.declarations.enumeration, 106  
pygccxml.declarations.free\_calldef, 108  
pygccxml.declarations.function\_traits, 114  
pygccxml.declarations.has\_operator\_matcher, 114  
pygccxml.declarations.location, 114  
pygccxml.declarations.matchers, 114  
pygccxml.declarations.mdecl\_wrapper, 115  
pygccxml.declarations.namespace, 116  
pygccxml.declarations.pattern\_parser, 120  
pygccxml.declarations.runtime\_errors, 121  
pygccxml.declarations.scopedef, 121  
pygccxml.declarations.smart\_pointer\_traits, 127  
pygccxml.declarations.templates, 127  
pygccxml.declarations.traits\_impl\_details, 129  
pygccxml.declarations.type\_traits, 129  
pygccxml.declarations.type\_traits\_classes, 131  
pygccxml.declarations.type\_visitor, 134  
pygccxml.declarations.typedef, 135  
pygccxml.declarations.variable, 136  
pygccxml.declarations.xml\_generators, 138  
pygccxml.parser, 138  
pygccxml.parser.config, 138  
pygccxml.parser.declarations\_cache, 141  
pygccxml.parser.declarations\_joiner, 142  
pygccxml.parser.directory\_cache, 142  
pygccxml.parser.etree\_scanner, 144  
pygccxml.parser.linker, 146  
pygccxml.parser.patcher, 148  
pygccxml.parser.project\_reader, 148  
pygccxml.parser.scanner, 150  
pygccxml.parser.source\_reader, 152  
pygccxml.utils, 153  
pygccxml.utils.utils, 154  
pygccxml.utils.xml\_generators, 156



## A

- access (hierarchy\_info\_t attribute), 75
- access() (ietree\_scanner\_t method), 144
- access() (scanner\_t method), 150
- access\_type (casting\_operator\_t attribute), 53
- access\_type (constructor\_t attribute), 55
- access\_type (declaration\_algs\_cache\_t attribute), 49
- access\_type (dependency\_info\_t attribute), 75
- access\_type (destructor\_t attribute), 57
- access\_type (hierarchy\_info\_t attribute), 75
- access\_type (member\_calldef\_t attribute), 59
- access\_type (member\_function\_t attribute), 61
- access\_type (member\_operator\_t attribute), 63
- access\_type (variable\_t attribute), 136
- access\_type\_matcher (in module pygccxml.declarations), 47
- access\_type\_matcher\_t (class in pygccxml.declarations.matchers), 114
- ACCESS\_TYPES (class in pygccxml.declarations.class\_declaration), 68
- acquire\_filename() (filename\_repository\_t method), 143
- adopt\_declaration() (class\_t method), 70
- adopt\_declaration() (namespace\_t method), 116
- aliases (class\_declaration\_t attribute), 69
- aliases (class\_t attribute), 70
- ALL (ACCESS\_TYPES attribute), 68
- all (CALLING\_CONVENTION\_TYPES attribute), 68
- ALL (CLASS\_TYPES attribute), 68
- ALL (VIRTUALITY\_TYPES attribute), 68
- ALL\_AT\_ONCE (COMPILATION\_MODE attribute), 148
- all\_container\_traits (in module pygccxml.declarations.container\_traits), 75
- all\_loggers (loggers attribute), 155
- ALLOW\_EMPTY\_MDECL\_WRAPPER (class\_t attribute), 70
- ALLOW\_EMPTY\_MDECL\_WRAPPER (namespace\_t attribute), 116
- ALLOW\_EMPTY\_MDECL\_WRAPPER (scopedef\_t attribute), 123
- and\_matcher (in module pygccxml.declarations), 47
- and\_matcher\_t (class in pygccxml.declarations.matchers), 114
- append\_cflags() (parser\_configuration\_t method), 139
- append\_cflags() (xml\_generator\_configuration\_t method), 140
- append\_value() (enumeration\_t method), 106
- apply\_visitor() (in module pygccxml.declarations.algorithm), 48
- args (declaration\_not\_found\_t attribute), 121
- args (multiple\_declarations\_found\_t attribute), 121
- args (visit\_function\_has\_not\_been\_found\_t attribute), 121
- args() (in module pygccxml.declarations.call\_invocation), 50
- args() (in module pygccxml.declarations.templates), 127
- args() (parser\_t method), 120
- argument\_t (class in pygccxml.declarations.calldef), 51
- argument\_types (calldef\_t attribute), 51
- argument\_types (casting\_operator\_t attribute), 53
- argument\_types (constructor\_t attribute), 55
- argument\_types (destructor\_t attribute), 57
- argument\_types (free\_calldef\_t attribute), 108
- argument\_types (free\_function\_t attribute), 110
- argument\_types (free\_operator\_t attribute), 112
- argument\_types (member\_calldef\_t attribute), 59
- argument\_types (member\_function\_t attribute), 61
- argument\_types (member\_operator\_t attribute), 64
- argument\_types (operator\_t attribute), 66
- arguments (calldef\_t attribute), 51
- arguments (casting\_operator\_t attribute), 53
- arguments (constructor\_t attribute), 55
- arguments (destructor\_t attribute), 58
- arguments (free\_calldef\_t attribute), 108
- arguments (free\_function\_t attribute), 110
- arguments (free\_operator\_t attribute), 112
- arguments (member\_calldef\_t attribute), 60
- arguments (member\_function\_t attribute), 62
- arguments (member\_operator\_t attribute), 64

arguments (operator\_t attribute), 66  
arguments\_types (calldef\_type\_t attribute), 78  
arguments\_types (free\_function\_type\_t attribute), 84  
arguments\_types (member\_function\_type\_t attribute), 92  
array\_item\_type() (in module pygccxml.declarations.type\_traits), 129  
array\_size() (in module pygccxml.declarations.type\_traits), 129  
array\_t (class in pygccxml.declarations.cpptypes), 77  
as\_tuple() (location\_t method), 114  
attributes (argument\_t attribute), 51  
attributes (calldef\_t attribute), 52  
attributes (casting\_operator\_t attribute), 53  
attributes (class\_declaration\_t attribute), 69  
attributes (class\_t attribute), 70  
attributes (constructor\_t attribute), 55  
attributes (declaration\_t attribute), 102  
attributes (destructor\_t attribute), 58  
attributes (enumeration\_t attribute), 106  
attributes (free\_calldef\_t attribute), 108  
attributes (free\_function\_t attribute), 110  
attributes (free\_operator\_t attribute), 112  
attributes (member\_calldef\_t attribute), 60  
attributes (member\_function\_t attribute), 62  
attributes (member\_operator\_t attribute), 64  
attributes (namespace\_t attribute), 116  
attributes (operator\_t attribute), 66  
attributes (scopedef\_t attribute), 123  
attributes (typedef\_t attribute), 135  
attributes (variable\_t attribute), 137  
auto\_ptr\_traits (class in pygccxml.declarations.smart\_pointer\_traits), 127

## B

base (array\_t attribute), 77  
base (compound\_t attribute), 80  
base (const\_t attribute), 81  
base (elaborated\_t attribute), 82  
base (member\_variable\_type\_t attribute), 93  
base (pointer\_t attribute), 93  
base (reference\_t attribute), 94  
base (restrict\_t attribute), 94  
base (volatile\_t attribute), 99  
base\_type() (in module pygccxml.declarations.type\_traits), 129  
bases (class\_t attribute), 70  
bind\_aliases() (in module pygccxml.parser.declarations\_joiner), 142  
bind\_aliases() (in module pygccxml.parser.source\_reader), 152  
bits (variable\_t attribute), 137  
bool\_t (class in pygccxml.declarations.cpptypes), 78  
build\_decl\_string() (array\_t method), 77  
build\_decl\_string() (bool\_t method), 78  
build\_decl\_string() (char\_t method), 78  
build\_decl\_string() (complex\_double\_t method), 79  
build\_decl\_string() (complex\_float\_t method), 79  
build\_decl\_string() (complex\_long\_double\_t method), 80  
build\_decl\_string() (compound\_t method), 80  
build\_decl\_string() (const\_t method), 81  
build\_decl\_string() (declared\_t method), 81  
build\_decl\_string() (double\_t method), 82  
build\_decl\_string() (dummy\_type\_t method), 82  
build\_decl\_string() (elaborated\_t method), 83  
build\_decl\_string() (ellipsis\_t method), 83  
build\_decl\_string() (float\_t method), 83  
build\_decl\_string() (free\_function\_type\_t method), 84  
build\_decl\_string() (fundamental\_t method), 85  
build\_decl\_string() (int128\_t method), 85  
build\_decl\_string() (int\_t method), 86  
build\_decl\_string() (java\_fundamental\_t method), 86  
build\_decl\_string() (jboolean\_t method), 86  
build\_decl\_string() (jbyte\_t method), 87  
build\_decl\_string() (jchar\_t method), 87  
build\_decl\_string() (jdouble\_t method), 88  
build\_decl\_string() (jfloat\_t method), 88  
build\_decl\_string() (jint\_t method), 89  
build\_decl\_string() (jlong\_t method), 89  
build\_decl\_string() (jshort\_t method), 89  
build\_decl\_string() (long\_double\_t method), 90  
build\_decl\_string() (long\_int\_t method), 90  
build\_decl\_string() (long\_long\_int\_t method), 91  
build\_decl\_string() (long\_long\_unsigned\_int\_t method), 91  
build\_decl\_string() (long\_unsigned\_int\_t method), 92  
build\_decl\_string() (member\_function\_type\_t method), 92  
build\_decl\_string() (member\_variable\_type\_t method), 93  
build\_decl\_string() (pointer\_t method), 94  
build\_decl\_string() (reference\_t method), 94  
build\_decl\_string() (restrict\_t method), 94  
build\_decl\_string() (short\_int\_t method), 95  
build\_decl\_string() (short\_unsigned\_int\_t method), 95  
build\_decl\_string() (signed\_char\_t method), 96  
build\_decl\_string() (type\_t method), 96  
build\_decl\_string() (uint128\_t method), 97  
build\_decl\_string() (unknown\_t method), 97  
build\_decl\_string() (unsigned\_char\_t method), 98  
build\_decl\_string() (unsigned\_int\_t method), 98  
build\_decl\_string() (void\_t method), 99  
build\_decl\_string() (volatile\_t method), 99  
build\_decl\_string() (wchar\_t method), 99  
byte\_align (array\_t attribute), 77  
byte\_align (bool\_t attribute), 78  
byte\_align (byte\_info attribute), 50  
byte\_align (char\_t attribute), 78  
byte\_align (class\_t attribute), 70

byte\_align (complex\_double\_t attribute), 79  
 byte\_align (complex\_float\_t attribute), 79  
 byte\_align (complex\_long\_double\_t attribute), 80  
 byte\_align (compound\_t attribute), 80  
 byte\_align (const\_t attribute), 81  
 byte\_align (declared\_t attribute), 81  
 byte\_align (double\_t attribute), 82  
 byte\_align (dummy\_type\_t attribute), 82  
 byte\_align (elaborated\_t attribute), 83  
 byte\_align (ellipsis\_t attribute), 83  
 byte\_align (enumeration\_t attribute), 106  
 byte\_align (float\_t attribute), 83  
 byte\_align (free\_function\_type\_t attribute), 84  
 byte\_align (fundamental\_t attribute), 85  
 byte\_align (int128\_t attribute), 85  
 byte\_align (int\_t attribute), 86  
 byte\_align (java\_fundamental\_t attribute), 86  
 byte\_align (jboolean\_t attribute), 86  
 byte\_align (jbyte\_t attribute), 87  
 byte\_align (jchar\_t attribute), 87  
 byte\_align (jdouble\_t attribute), 88  
 byte\_align (jfloat\_t attribute), 88  
 byte\_align (jint\_t attribute), 89  
 byte\_align (jlong\_t attribute), 89  
 byte\_align (jshort\_t attribute), 89  
 byte\_align (long\_double\_t attribute), 90  
 byte\_align (long\_int\_t attribute), 90  
 byte\_align (long\_long\_int\_t attribute), 91  
 byte\_align (long\_long\_unsigned\_int\_t attribute), 91  
 byte\_align (long\_unsigned\_int\_t attribute), 92  
 byte\_align (member\_function\_type\_t attribute), 92  
 byte\_align (member\_variable\_type\_t attribute), 93  
 byte\_align (pointer\_t attribute), 94  
 byte\_align (reference\_t attribute), 94  
 byte\_align (restrict\_t attribute), 94  
 byte\_align (short\_int\_t attribute), 95  
 byte\_align (short\_unsigned\_int\_t attribute), 95  
 byte\_align (signed\_char\_t attribute), 96  
 byte\_align (type\_t attribute), 96  
 byte\_align (typedef\_t attribute), 135  
 byte\_align (uint128\_t attribute), 97  
 byte\_align (unknown\_t attribute), 97  
 byte\_align (unsigned\_char\_t attribute), 98  
 byte\_align (unsigned\_int\_t attribute), 98  
 byte\_align (void\_t attribute), 99  
 byte\_align (volatile\_t attribute), 99  
 byte\_align (wchar\_t attribute), 99  
 byte\_info (class in pygccxml.declarations.byte\_info), 50  
 byte\_offset (variable\_t attribute), 137  
 byte\_size (array\_t attribute), 77  
 byte\_size (bool\_t attribute), 78  
 byte\_size (byte\_info attribute), 50  
 byte\_size (char\_t attribute), 79  
 byte\_size (class\_t attribute), 70

byte\_size (complex\_double\_t attribute), 79  
 byte\_size (complex\_float\_t attribute), 79  
 byte\_size (complex\_long\_double\_t attribute), 80  
 byte\_size (compound\_t attribute), 80  
 byte\_size (const\_t attribute), 81  
 byte\_size (declared\_t attribute), 81  
 byte\_size (double\_t attribute), 82  
 byte\_size (dummy\_type\_t attribute), 82  
 byte\_size (elaborated\_t attribute), 83  
 byte\_size (ellipsis\_t attribute), 83  
 byte\_size (enumeration\_t attribute), 106  
 byte\_size (float\_t attribute), 84  
 byte\_size (free\_function\_type\_t attribute), 84  
 byte\_size (fundamental\_t attribute), 85  
 byte\_size (int128\_t attribute), 85  
 byte\_size (int\_t attribute), 86  
 byte\_size (java\_fundamental\_t attribute), 86  
 byte\_size (jboolean\_t attribute), 86  
 byte\_size (jbyte\_t attribute), 87  
 byte\_size (jchar\_t attribute), 87  
 byte\_size (jdouble\_t attribute), 88  
 byte\_size (jfloat\_t attribute), 88  
 byte\_size (jint\_t attribute), 89  
 byte\_size (jlong\_t attribute), 89  
 byte\_size (jshort\_t attribute), 90  
 byte\_size (long\_double\_t attribute), 90  
 byte\_size (long\_int\_t attribute), 90  
 byte\_size (long\_long\_int\_t attribute), 91  
 byte\_size (long\_long\_unsigned\_int\_t attribute), 91  
 byte\_size (long\_unsigned\_int\_t attribute), 92  
 byte\_size (member\_function\_type\_t attribute), 92  
 byte\_size (member\_variable\_type\_t attribute), 93  
 byte\_size (pointer\_t attribute), 94  
 byte\_size (reference\_t attribute), 94  
 byte\_size (restrict\_t attribute), 95  
 byte\_size (short\_int\_t attribute), 95  
 byte\_size (short\_unsigned\_int\_t attribute), 95  
 byte\_size (signed\_char\_t attribute), 96  
 byte\_size (type\_t attribute), 96  
 byte\_size (typedef\_t attribute), 135  
 byte\_size (uint128\_t attribute), 97  
 byte\_size (unknown\_t attribute), 97  
 byte\_size (unsigned\_char\_t attribute), 98  
 byte\_size (unsigned\_int\_t attribute), 98  
 byte\_size (void\_t attribute), 99  
 byte\_size (volatile\_t attribute), 99  
 byte\_size (wchar\_t attribute), 100

## C

cache (calldef\_t attribute), 52  
 cache (casting\_operator\_t attribute), 53  
 cache (class\_declaration\_t attribute), 69  
 cache (class\_t attribute), 70  
 cache (constructor\_t attribute), 56

cache (declaration\_t attribute), 102  
cache (destructor\_t attribute), 58  
cache (enumeration\_t attribute), 106  
cache (free\_calldef\_t attribute), 108  
cache (free\_function\_t attribute), 110  
cache (free\_operator\_t attribute), 112  
cache (member\_calldef\_t attribute), 60  
cache (member\_function\_t attribute), 62  
cache (member\_operator\_t attribute), 64  
cache (namespace\_t attribute), 116  
cache (operator\_t attribute), 66  
cache (scopedef\_t attribute), 123  
cache (typedef\_t attribute), 135  
cache (variable\_t attribute), 137  
cache\_base\_t (class in pygccxml.parser.declarations\_cache), 141  
cached (class in pygccxml.utils.utils), 154  
cached\_source\_file (file\_configuration\_t attribute), 149  
CACHED\_SOURCE\_FILE (file\_configuration\_t.CONTENT\_TYPE attribute), 149  
cached\_value() (cache\_base\_t method), 141  
cached\_value() (directory\_cache\_t method), 143  
cached\_value() (dummy\_cache\_t method), 141  
cached\_value() (file\_cache\_t method), 142  
call\_redirector\_t (class in pygccxml.declarations.mdecl\_wrapper), 115  
calldef() (class\_t method), 71  
calldef() (namespace\_t method), 116  
calldef() (scopedef\_t method), 124  
calldef\_matcher (in module pygccxml.declarations), 47  
calldef\_matcher\_t (class in pygccxml.declarations.declarations\_matchers), 104  
calldef\_t (class in pygccxml.declarations.calldef), 51  
calldef\_type\_t (class in pygccxml.declarations.cpptypes), 78  
calldefs() (class\_t method), 71  
calldefs() (ietree\_scanner\_t method), 144  
calldefs() (namespace\_t method), 116  
calldefs() (scanner\_t method), 150  
calldefs() (scopedef\_t method), 124  
calling\_convention (calldef\_t attribute), 52  
calling\_convention (casting\_operator\_t attribute), 54  
calling\_convention (constructor\_t attribute), 56  
calling\_convention (destructor\_t attribute), 58  
calling\_convention (free\_calldef\_t attribute), 108  
calling\_convention (free\_function\_t attribute), 110  
calling\_convention (free\_operator\_t attribute), 112  
calling\_convention (member\_calldef\_t attribute), 60  
calling\_convention (member\_function\_t attribute), 62  
calling\_convention (member\_operator\_t attribute), 64  
calling\_convention (operator\_t attribute), 66  
CALLING\_CONVENTION\_TYPES (class in pygccxml.declarations.calldef\_types), 67  
casting\_operator() (class\_t method), 71  
casting\_operator() (namespace\_t method), 116  
casting\_operator() (scopedef\_t method), 124  
casting\_operator\_patcher\_t (class in pygccxml.parser.patcher), 148  
casting\_operator\_t (class in pygccxml.declarations.calldef\_members), 53  
casting\_operators() (class\_t method), 71  
casting\_operators() (namespace\_t method), 116  
casting\_operators() (scopedef\_t method), 124  
castxml\_epic\_version (parser\_configuration\_t attribute), 139  
castxml\_epic\_version (xml\_generator\_configuration\_t attribute), 140  
CDECL (CALLING\_CONVENTION\_TYPES attribute), 67  
cflags (parser\_configuration\_t attribute), 139  
cflags (xml\_generator\_configuration\_t attribute), 140  
char\_t (class in pygccxml.declarations.cpptypes), 78  
characters() (ietree\_scanner\_t method), 144  
characters() (scanner\_t method), 150  
check\_name() (calldef\_matcher\_t method), 104  
check\_name() (declaration\_matcher\_t method), 105  
check\_name() (namespace\_matcher\_t method), 105  
check\_name() (operator\_matcher\_t method), 105  
check\_name() (variable\_matcher\_t method), 105  
CLASS (CLASS\_TYPES attribute), 68  
class\_() (class\_t method), 71  
class\_() (namespace\_t method), 116  
class\_() (scopedef\_t method), 124  
class\_declaration() (container\_traits\_impl\_t method), 76  
class\_declaration\_t (class in pygccxml.declarations.class\_declaration), 69  
class\_declaration\_traits (in module pygccxml.declarations.type\_traits\_classes), 131  
class\_inst (member\_function\_type\_t attribute), 92  
class\_t (class in pygccxml.declarations.class\_declaration), 70  
class\_traits (in module pygccxml.declarations.type\_traits\_classes), 131  
class\_type (class\_t attribute), 71  
CLASS\_TYPES (class in pygccxml.declarations.class\_declaration), 68  
class\_types (free\_operator\_t attribute), 112  
classes() (class\_t method), 71  
classes() (namespace\_t method), 116  
classes() (scopedef\_t method), 124  
clear\_optimizer() (class\_t method), 71  
clear\_optimizer() (namespace\_t method), 117  
clear\_optimizer() (scopedef\_t method), 124  
clone() (argument\_t method), 51  
clone() (array\_t method), 78



- clone() (bool\_t method), 78
- clone() (char\_t method), 79
- clone() (complex\_double\_t method), 79
- clone() (complex\_float\_t method), 80
- clone() (complex\_long\_double\_t method), 80
- clone() (compound\_t method), 80
- clone() (const\_t method), 81
- clone() (decl\_printer\_t method), 101
- clone() (declared\_t method), 81
- clone() (double\_t method), 82
- clone() (dummy\_type\_t method), 82
- clone() (elaborated\_t method), 83
- clone() (ellipsis\_t method), 83
- clone() (float\_t method), 84
- clone() (free\_function\_type\_t method), 84
- clone() (fundamental\_t method), 85
- clone() (int128\_t method), 85
- clone() (int\_t method), 86
- clone() (java\_fundamental\_t method), 86
- clone() (jboolean\_t method), 87
- clone() (jbyte\_t method), 87
- clone() (jchar\_t method), 87
- clone() (jdouble\_t method), 88
- clone() (jfloat\_t method), 88
- clone() (jint\_t method), 89
- clone() (jlong\_t method), 89
- clone() (jshort\_t method), 90
- clone() (long\_double\_t method), 90
- clone() (long\_int\_t method), 91
- clone() (long\_long\_int\_t method), 91
- clone() (long\_long\_unsigned\_int\_t method), 91
- clone() (long\_unsigned\_int\_t method), 92
- clone() (member\_function\_type\_t method), 93
- clone() (member\_variable\_type\_t method), 93
- clone() (parser\_configuration\_t method), 139
- clone() (pointer\_t method), 94
- clone() (reference\_t method), 94
- clone() (restrict\_t method), 95
- clone() (short\_int\_t method), 95
- clone() (short\_unsigned\_int\_t method), 96
- clone() (signed\_char\_t method), 96
- clone() (type\_t method), 97
- clone() (uint128\_t method), 97
- clone() (unknown\_t method), 97
- clone() (unsigned\_char\_t method), 98
- clone() (unsigned\_int\_t method), 98
- clone() (void\_t method), 99
- clone() (volatile\_t method), 99
- clone() (wchar\_t method), 100
- clone() (xml\_generator\_configuration\_t method), 140
- cmp\_data (declaration\_algs\_cache\_t attribute), 49
- COMPILATION\_MODE (class in pygccxml.parser.project\_reader), 148
- compiler (parser\_configuration\_t attribute), 139
- compiler (xml\_generator\_configuration\_t attribute), 140
- compiler\_path (parser\_configuration\_t attribute), 139
- compiler\_path (xml\_generator\_configuration\_t attribute), 140
- complex\_double\_t (class in pygccxml.declarations.cpptypes), 79
- complex\_float\_t (class in pygccxml.declarations.cpptypes), 79
- complex\_long\_double\_t (class in pygccxml.declarations.cpptypes), 80
- compound\_t (class in pygccxml.declarations.cpptypes), 80
- config\_signature (record\_t attribute), 142
- configuration\_signature() (in module pygccxml.parser.declarations\_cache), 141
- const\_t (class in pygccxml.declarations.cpptypes), 81
- constructor() (class\_t method), 71
- constructor() (namespace\_t method), 117
- constructor() (scoped\_def\_t method), 124
- constructor\_t (class in pygccxml.declarations.calldef\_members), 55
- constructors() (class\_t method), 71
- constructors() (namespace\_t method), 117
- constructors() (scoped\_def\_t method), 124
- container\_element\_type (declaration\_algs\_cache\_t attribute), 49
- container\_key\_type (declaration\_algs\_cache\_t attribute), 49
- container\_traits (declaration\_algs\_cache\_t attribute), 49
- container\_traits\_impl\_t (class in pygccxml.declarations.container\_traits), 76
- contains\_parent\_dir() (in module pygccxml.utils.utils), 154
- content\_type (file\_configuration\_t attribute), 149
- CPPNAME (bool\_t attribute), 78
- CPPNAME (char\_t attribute), 78
- CPPNAME (complex\_double\_t attribute), 79
- CPPNAME (complex\_float\_t attribute), 79
- CPPNAME (complex\_long\_double\_t attribute), 80
- CPPNAME (double\_t attribute), 82
- CPPNAME (float\_t attribute), 83
- CPPNAME (int128\_t attribute), 85
- CPPNAME (int\_t attribute), 85
- CPPNAME (long\_double\_t attribute), 90
- CPPNAME (long\_int\_t attribute), 90
- CPPNAME (long\_long\_int\_t attribute), 91
- CPPNAME (long\_long\_unsigned\_int\_t attribute), 91
- CPPNAME (long\_unsigned\_int\_t attribute), 92
- CPPNAME (short\_int\_t attribute), 95
- CPPNAME (short\_unsigned\_int\_t attribute), 95
- CPPNAME (signed\_char\_t attribute), 96
- CPPNAME (uint128\_t attribute), 97
- CPPNAME (unsigned\_char\_t attribute), 98
- CPPNAME (unsigned\_int\_t attribute), 98

CPPNAME (void\_t attribute), 98  
CPPNAME (wchar\_t attribute), 99  
create\_cached\_source\_fc() (in module pygccxml.parser.project\_reader), 148  
create\_casting\_operator() (decl\_factory\_t method), 100  
create\_class() (decl\_factory\_t method), 100  
create\_class\_declaration() (decl\_factory\_t method), 100  
create\_compiler\_path() (in module pygccxml.parser.config), 138  
create\_constructor() (decl\_factory\_t method), 100  
create\_decl\_string() (calldef\_t method), 52  
create\_decl\_string() (casting\_operator\_t method), 54  
create\_decl\_string() (class\_declaration\_t method), 69  
create\_decl\_string() (class\_t method), 71  
create\_decl\_string() (constructor\_t method), 56  
create\_decl\_string() (declaration\_t method), 103  
create\_decl\_string() (destructor\_t method), 58  
create\_decl\_string() (enumeration\_t method), 107  
create\_decl\_string() (free\_calldef\_t method), 108  
create\_decl\_string() (free\_function\_t method), 110  
create\_decl\_string() (free\_function\_type\_t static method), 84  
create\_decl\_string() (free\_operator\_t method), 112  
create\_decl\_string() (member\_calldef\_t method), 60  
create\_decl\_string() (member\_function\_t method), 62  
create\_decl\_string() (member\_function\_type\_t static method), 93  
create\_decl\_string() (member\_operator\_t method), 64  
create\_decl\_string() (namespace\_t method), 117  
create\_decl\_string() (operator\_t method), 66  
create\_decl\_string() (scopeddef\_t method), 124  
create\_decl\_string() (typedef\_t method), 135  
create\_decl\_string() (variable\_t method), 137  
create\_destructor() (decl\_factory\_t method), 100  
create\_enumeration() (decl\_factory\_t method), 100  
create\_free\_function() (decl\_factory\_t method), 100  
create\_free\_operator() (decl\_factory\_t method), 100  
create\_gccxml\_fc() (in module pygccxml.parser.project\_reader), 148  
create\_key() (record\_t static method), 142  
create\_member\_function() (decl\_factory\_t method), 100  
create\_member\_operator() (decl\_factory\_t method), 100  
create\_namespace() (decl\_factory\_t method), 100  
create\_source\_fc() (in module pygccxml.parser.project\_reader), 149  
create\_temp\_file\_name() (in module pygccxml.utils.utils), 154  
create\_text\_fc() (in module pygccxml.parser.project\_reader), 149  
create\_typedef() (decl\_factory\_t method), 100  
create\_typedef() (free\_function\_type\_t method), 84  
create\_typedef() (member\_function\_type\_t method), 93  
create\_variable() (decl\_factory\_t method), 100  
create\_xml\_file() (source\_reader\_t method), 153

create\_xml\_file\_from\_string() (source\_reader\_t method), 153  
custom\_matcher (in module pygccxml.declarations), 47  
custom\_matcher\_t (class in pygccxml.declarations.matchers), 114  
cxx\_parser (loggers attribute), 155  
cxx\_standard (class in pygccxml.utils.utils), 154

## D

data (file\_configuration\_t attribute), 149  
dec\_ref\_count() (filename\_entry\_t method), 143  
decl (dependency\_info\_t attribute), 75  
decl() (class\_t method), 71  
decl() (namespace\_t method), 117  
decl() (scopeddef\_t method), 124  
decl\_factory\_t (class in pygccxml.declarations.decl\_factory), 100  
decl\_name\_only (calldef\_matcher\_t attribute), 105  
decl\_name\_only (declaration\_matcher\_t attribute), 105  
decl\_name\_only (namespace\_matcher\_t attribute), 105  
decl\_name\_only (operator\_matcher\_t attribute), 105  
decl\_name\_only (variable\_matcher\_t attribute), 105  
decl\_printer\_t (class in pygccxml.declarations.decl\_printer), 101  
decl\_string (array\_t attribute), 78  
decl\_string (bool\_t attribute), 78  
decl\_string (calldef\_t attribute), 52  
decl\_string (casting\_operator\_t attribute), 54  
decl\_string (char\_t attribute), 79  
decl\_string (class\_declaration\_t attribute), 69  
decl\_string (class\_t attribute), 71  
decl\_string (complex\_double\_t attribute), 79  
decl\_string (complex\_float\_t attribute), 80  
decl\_string (complex\_long\_double\_t attribute), 80  
decl\_string (compound\_t attribute), 81  
decl\_string (const\_t attribute), 81  
decl\_string (constructor\_t attribute), 56  
decl\_string (declared\_t attribute), 81  
decl\_string (declaration\_t attribute), 103  
decl\_string (destructor\_t attribute), 58  
decl\_string (double\_t attribute), 82  
decl\_string (dummy\_type\_t attribute), 82  
decl\_string (elaborated\_t attribute), 83  
decl\_string (ellipsis\_t attribute), 83  
decl\_string (enumeration\_t attribute), 107  
decl\_string (float\_t attribute), 84  
decl\_string (free\_calldef\_t attribute), 108  
decl\_string (free\_function\_t attribute), 110  
decl\_string (free\_function\_type\_t attribute), 84  
decl\_string (free\_operator\_t attribute), 112  
decl\_string (fundamental\_t attribute), 85  
decl\_string (int128\_t attribute), 85  
decl\_string (int\_t attribute), 86  
decl\_string (java\_fundamental\_t attribute), 86



decl\_string (jboolean\_t attribute), 87  
 decl\_string (jbyte\_t attribute), 87  
 decl\_string (jchar\_t attribute), 88  
 decl\_string (jdouble\_t attribute), 88  
 decl\_string (jfloat\_t attribute), 88  
 decl\_string (jint\_t attribute), 89  
 decl\_string (jlong\_t attribute), 89  
 decl\_string (jshort\_t attribute), 90  
 decl\_string (long\_double\_t attribute), 90  
 decl\_string (long\_int\_t attribute), 91  
 decl\_string (long\_long\_int\_t attribute), 91  
 decl\_string (long\_long\_unsigned\_int\_t attribute), 92  
 decl\_string (long\_unsigned\_int\_t attribute), 92  
 decl\_string (member\_calldef\_t attribute), 60  
 decl\_string (member\_function\_t attribute), 62  
 decl\_string (member\_function\_type\_t attribute), 93  
 decl\_string (member\_operator\_t attribute), 64  
 decl\_string (member\_variable\_type\_t attribute), 93  
 decl\_string (namespace\_t attribute), 117  
 decl\_string (operator\_t attribute), 66  
 decl\_string (pointer\_t attribute), 94  
 decl\_string (reference\_t attribute), 94  
 decl\_string (restrict\_t attribute), 95  
 decl\_string (scopedef\_t attribute), 124  
 decl\_string (short\_int\_t attribute), 95  
 decl\_string (short\_unsigned\_int\_t attribute), 96  
 decl\_string (signed\_char\_t attribute), 96  
 decl\_string (type\_algs\_cache\_t attribute), 49  
 decl\_string (type\_t attribute), 97  
 decl\_string (typedef\_t attribute), 135  
 decl\_string (uint128\_t attribute), 97  
 decl\_string (unknown\_t attribute), 97  
 decl\_string (unsigned\_char\_t attribute), 98  
 decl\_string (unsigned\_int\_t attribute), 98  
 decl\_string (variable\_t attribute), 137  
 decl\_string (void\_t attribute), 99  
 decl\_string (volatile\_t attribute), 99  
 decl\_string (wchar\_t attribute), 100  
 decl\_type (argument\_t attribute), 51  
 decl\_type (typedef\_t attribute), 135  
 decl\_type (variable\_t attribute), 137  
 decl\_visitor\_t (class in pygccxml.declarations.decl\_visitor), 102  
 declared\_t (class in pygccxml.declarations.cpptypes), 81  
 declaration (declared\_t attribute), 81  
 declaration (dependency\_info\_t attribute), 75  
 declaration\_algs\_cache\_t (class in pygccxml.declarations.algorithms\_cache), 49  
 declaration\_files() (in module pygccxml.declarations.scopedef), 121  
 declaration\_matcher (in module pygccxml.declarations), 47  
 declaration\_matcher\_t (class in pygccxml.declarations.declarations\_matchers), 105  
 declaration\_not\_found\_t, 121  
 declaration\_not\_found\_t (class\_t attribute), 71  
 declaration\_not\_found\_t (namespace\_t attribute), 117  
 declaration\_not\_found\_t (scopedef\_t attribute), 124  
 declaration\_path (declaration\_algs\_cache\_t attribute), 49  
 declaration\_path (hierarchy\_info\_t attribute), 75  
 declaration\_path() (in module pygccxml.declarations.declaration\_utils), 104  
 declaration\_path\_hash (hierarchy\_info\_t attribute), 75  
 declaration\_t (class in pygccxml.declarations.declaration), 102  
 declaration\_xxx\_traits (class in pygccxml.declarations.type\_traits\_classes), 131  
 declarations (class\_t attribute), 71  
 declarations (namespace\_t attribute), 117  
 declarations (record\_t attribute), 142  
 declarations (scopedef\_t attribute), 124  
 declarations() (ietree\_scanner\_t method), 144  
 declarations() (scanner\_t method), 150  
 declarations\_cache (loggers attribute), 155  
 decls() (class\_t method), 71  
 decls() (namespace\_t method), 117  
 decls() (scopedef\_t method), 124  
 decompose\_class() (in module pygccxml.declarations.type\_traits), 129  
 decompose\_type() (in module pygccxml.declarations.type\_traits), 129  
 decorated\_call\_prefix() (defaults\_eraser method), 76  
 decorated\_call\_suffix() (defaults\_eraser method), 76  
 decorated\_name (calldef\_t attribute), 52  
 decorated\_name (casting\_operator\_t attribute), 54  
 decorated\_name (class\_declaration\_t attribute), 69  
 decorated\_name (class\_t attribute), 71  
 decorated\_name (constructor\_t attribute), 56  
 decorated\_name (declaration\_t attribute), 103  
 decorated\_name (destructor\_t attribute), 58  
 decorated\_name (enumeration\_t attribute), 107  
 decorated\_name (free\_calldef\_t attribute), 108  
 decorated\_name (free\_function\_t attribute), 110  
 decorated\_name (free\_operator\_t attribute), 112  
 decorated\_name (member\_calldef\_t attribute), 60  
 decorated\_name (member\_function\_t attribute), 62  
 decorated\_name (member\_operator\_t attribute), 64  
 decorated\_name (namespace\_t attribute), 117  
 decorated\_name (operator\_t attribute), 66  
 decorated\_name (scopedef\_t attribute), 124  
 decorated\_name (typedef\_t attribute), 135  
 decorated\_name (variable\_t attribute), 137  
 default\_argument\_patcher\_t (class in pygccxml.parser.patcher), 148  
 default\_value (argument\_t attribute), 51

defaults\_eraser (class in pygccxml.declarations.container\_traits), 76

define\_symbols (parser\_configuration\_t attribute), 139

define\_symbols (xml\_generator\_configuration\_t attribute), 140

deleter() (cached method), 154

demangled (calldef\_t attribute), 52

demangled (casting\_operator\_t attribute), 54

demangled (class\_declaration\_t attribute), 69

demangled (class\_t attribute), 71

demangled (constructor\_t attribute), 56

demangled (declaration\_t attribute), 103

demangled (destructor\_t attribute), 58

demangled (enumeration\_t attribute), 107

demangled (free\_calldef\_t attribute), 108

demangled (free\_function\_t attribute), 110

demangled (free\_operator\_t attribute), 112

demangled (member\_calldef\_t attribute), 60

demangled (member\_function\_t attribute), 62

demangled (member\_operator\_t attribute), 64

demangled (namespace\_t attribute), 117

demangled (operator\_t attribute), 66

demangled (scopedef\_t attribute), 124

demangled (typedef\_t attribute), 136

demangled (variable\_t attribute), 137

demangled\_name (calldef\_t attribute), 52

demangled\_name (casting\_operator\_t attribute), 54

demangled\_name (constructor\_t attribute), 56

demangled\_name (declaration\_algs\_cache\_t attribute), 49

demangled\_name (destructor\_t attribute), 58

demangled\_name (free\_calldef\_t attribute), 108

demangled\_name (free\_function\_t attribute), 110

demangled\_name (free\_operator\_t attribute), 112

demangled\_name (member\_calldef\_t attribute), 60

demangled\_name (member\_function\_t attribute), 62

demangled\_name (member\_operator\_t attribute), 64

demangled\_name (operator\_t attribute), 66

depend\_on\_it (dependency\_info\_t attribute), 75

dependency\_info\_t (class in pygccxml.declarations.class\_declaration), 75

DeprecationWrapper (class in pygccxml.utils.utils), 154

derived (class\_t attribute), 72

destructor\_t (class in pygccxml.declarations.calldef\_members), 57

dig\_declarations() (impl\_details static method), 75

directory\_cache\_t (class in pygccxml.parser.directory\_cache), 142

disable() (declaration\_algs\_cache\_t method), 49

disable() (type\_algs\_cache\_t static method), 49

does\_match\_definition() (in module pygccxml.declarations.type\_traits), 129

does\_match\_exist() (match\_declaration\_t method), 49

does\_throw (calldef\_t attribute), 52

does\_throw (casting\_operator\_t attribute), 54

does\_throw (constructor\_t attribute), 56

does\_throw (destructor\_t attribute), 58

does\_throw (free\_calldef\_t attribute), 108

does\_throw (free\_function\_t attribute), 110

does\_throw (free\_operator\_t attribute), 112

does\_throw (member\_calldef\_t attribute), 60

does\_throw (member\_function\_t attribute), 62

does\_throw (member\_operator\_t attribute), 64

does\_throw (operator\_t attribute), 66

double\_t (class in pygccxml.declarations.cpptypes), 82

dummy\_cache\_t (class in pygccxml.parser.declarations\_cache), 141

dummy\_type\_t (class in pygccxml.declarations.cpptypes), 82

dump\_declarations() (in module pygccxml.declarations.decl\_printer), 101

## E

elaborated\_info (class in pygccxml.declarations.elaborated\_info), 106

elaborated\_t (class in pygccxml.declarations.cpptypes), 82

elaborated\_type\_specifier (class\_t attribute), 72

elaborated\_type\_specifier (elaborated\_info attribute), 106

elaborated\_type\_specifier (enumeration\_t attribute), 107

element\_type() (container\_traits\_impl\_t method), 76

ellipsis (argument\_t attribute), 51

ellipsis\_t (class in pygccxml.declarations.cpptypes), 83

enable() (declaration\_algs\_cache\_t method), 49

enable() (type\_algs\_cache\_t static method), 49

enabled (declaration\_algs\_cache\_t attribute), 49

enabled (type\_algs\_cache\_t attribute), 49

endDocument() (ietree\_scanner\_t method), 144

endDocument() (scanner\_t method), 150

endElement() (ietree\_scanner\_t method), 144

endElement() (scanner\_t method), 150

endElementNS() (ietree\_scanner\_t method), 144

endElementNS() (scanner\_t method), 150

endPrefixMapping() (ietree\_scanner\_t method), 144

endPrefixMapping() (scanner\_t method), 151

enum() (class\_t method), 72

enum() (namespace\_t method), 117

enum() (scopedef\_t method), 124

enum\_declaration (in module pygccxml.declarations.type\_traits\_classes), 131

enum\_traits (in module pygccxml.declarations.type\_traits\_classes), 131

enumeration() (class\_t method), 72

enumeration() (namespace\_t method), 117

enumeration() (scopedef\_t method), 125

enumeration\_t (class in pygccxml.declarations.enumeration), 106

enumerations() (class\_t method), 72

enumerations() (namespace\_t method), 117

- enumerations() (scopedef\_t method), 125
  - enums() (class\_t method), 72
  - enums() (ietree\_scanner\_t method), 144
  - enums() (namespace\_t method), 117
  - enums() (scanner\_t method), 151
  - enums() (scopedef\_t method), 125
  - erase\_allocator() (defaults\_eraser method), 76
  - erase\_call() (defaults\_eraser method), 76
  - erase\_compare\_allocator() (defaults\_eraser method), 76
  - erase\_container() (defaults\_eraser method), 76
  - erase\_container\_compare() (defaults\_eraser method), 76
  - erase\_hash\_allocator() (defaults\_eraser method), 77
  - erase\_hashmap\_compare\_allocator() (defaults\_eraser method), 77
  - erase\_map\_compare\_allocator() (defaults\_eraser method), 77
  - erase\_recursive() (defaults\_eraser method), 77
  - exceptions (calldef\_t attribute), 52
  - exceptions (casting\_operator\_t attribute), 54
  - exceptions (constructor\_t attribute), 56
  - exceptions (destructor\_t attribute), 58
  - exceptions (free\_calldef\_t attribute), 109
  - exceptions (free\_function\_t attribute), 110
  - exceptions (free\_operator\_t attribute), 112
  - exceptions (member\_calldef\_t attribute), 60
  - exceptions (member\_function\_t attribute), 62
  - exceptions (member\_operator\_t attribute), 64
  - exceptions (operator\_t attribute), 66
  - explicit (constructor\_t attribute), 56
  - extract() (CALLING\_CONVENTION\_TYPES static method), 68
- ## F
- FASTCALL (CALLING\_CONVENTION\_TYPES attribute), 68
  - fdel (cached attribute), 154
  - fget (cached attribute), 154
  - FILE\_BY\_FILE (COMPILATION\_MODE attribute), 148
  - file\_cache\_t (class in pygccxml.parser.declarations\_cache), 141
  - file\_configuration\_t (class in pygccxml.parser.project\_reader), 149
  - file\_configuration\_t.CONTENT\_TYPE (class in pygccxml.parser.project\_reader), 149
  - file\_name (location\_t attribute), 114
  - file\_signature() (in module pygccxml.parser.declarations\_cache), 142
  - filename\_entry\_t (class in pygccxml.parser.directory\_cache), 143
  - filename\_repository\_t (class in pygccxml.parser.directory\_cache), 143
  - files() (ietree\_scanner\_t method), 144
  - files() (scanner\_t method), 151
  - find() (matcher static method), 122
  - find\_all\_declarations() (in module pygccxml.declarations.scopedef), 121
  - find\_args() (in module pygccxml.declarations.call\_invocation), 50
  - find\_args() (parser\_t method), 120
  - find\_container\_traits() (in module pygccxml.declarations.container\_traits), 77
  - find\_copy\_constructor() (in module pygccxml.declarations.type\_traits\_classes), 131
  - find\_declaration() (in module pygccxml.declarations.scopedef), 122
  - find\_first\_declaration() (in module pygccxml.declarations.scopedef), 122
  - find\_noncopyable\_vars() (in module pygccxml.declarations.type\_traits\_classes), 132
  - find\_out\_depend\_on\_it\_declarations() (dependency\_info\_t method), 75
  - find\_out\_member\_access\_type() (class\_t method), 72
  - find\_single() (matcher static method), 122
  - find\_trivial\_constructor() (in module pygccxml.declarations.type\_traits\_classes), 132
  - find\_value\_type() (impl\_details static method), 129
  - find\_xml\_generator() (in module pygccxml.utils.utils), 155
  - fix\_calldef\_decls() (in module pygccxml.parser.patcher), 148
  - flags (parser\_configuration\_t attribute), 139
  - flags (xml\_generator\_configuration\_t attribute), 140
  - float\_t (class in pygccxml.declarations.cpptypes), 83
  - flush() (cache\_base\_t method), 141
  - flush() (directory\_cache\_t method), 143
  - flush() (dummy\_cache\_t method), 141
  - flush() (file\_cache\_t method), 142
  - free\_calldef\_t (class in pygccxml.declarations.free\_calldef), 108
  - free\_fun() (namespace\_t method), 117
  - free\_function() (namespace\_t method), 118
  - free\_function\_t (class in pygccxml.declarations.free\_calldef), 110
  - free\_function\_type\_t (class in pygccxml.declarations.cpptypes), 84
  - free\_functions() (namespace\_t method), 118
  - free\_funs() (namespace\_t method), 118
  - free\_operator() (namespace\_t method), 118
  - free\_operator\_t (class in pygccxml.declarations.free\_calldef), 111
  - free\_operators() (namespace\_t method), 118
  - fset (cached attribute), 154
  - full\_name (declaration\_algs\_cache\_t attribute), 49
  - full\_name() (in module pygccxml.declarations.declaration\_utils), 104
  - full\_name\_from\_declaration\_path() (in module pygccxml.declarations.declaration\_utils), 104

full\_partial\_name (declaration\_algs\_cache\_t attribute), 49

function\_type() (casting\_operator\_t method), 54

function\_type() (constructor\_t method), 56

function\_type() (destructor\_t method), 58

function\_type() (free\_calldef\_t method), 109

function\_type() (free\_function\_t method), 110

function\_type() (free\_operator\_t method), 112

function\_type() (member\_calldef\_t method), 60

function\_type() (member\_function\_t method), 62

function\_type() (member\_operator\_t method), 64

FUNCTION\_VIRTUALITY\_TYPES (in module pygccxml.declarations.calldef\_types), 68

fundamental\_t (class in pygccxml.declarations.cpptypes), 85

FUNDAMENTAL\_TYPES (in module pygccxml.declarations.cpptypes), 77

## G

### GCCXML\_GENERATED\_FILE

(file\_configuration\_t.CONTENT\_TYPE attribute), 149

get\_architecture() (in module pygccxml.utils.utils), 155

get\_by\_name() (internal\_type\_traits static method), 127

get\_container\_or\_none() (container\_traits\_impl\_t method), 76

get\_declaration() (declaration\_xxx\_traits method), 131

get\_global\_namespace() (in module pygccxml.declarations.namespace), 116

get\_mangled\_name() (calldef\_t method), 52

get\_mangled\_name() (casting\_operator\_t method), 54

get\_mangled\_name() (class\_declaration\_t method), 69

get\_mangled\_name() (class\_t method), 72

get\_mangled\_name() (constructor\_t method), 56

get\_mangled\_name() (declaration\_t method), 103

get\_mangled\_name() (destructor\_t method), 58

get\_mangled\_name() (enumeration\_t method), 107

get\_mangled\_name() (free\_calldef\_t method), 109

get\_mangled\_name() (free\_function\_t method), 110

get\_mangled\_name() (free\_operator\_t method), 112

get\_mangled\_name() (member\_calldef\_t method), 60

get\_mangled\_name() (member\_function\_t method), 62

get\_mangled\_name() (member\_operator\_t method), 64

get\_mangled\_name() (namespace\_t method), 118

get\_mangled\_name() (operator\_t method), 66

get\_mangled\_name() (scopeddef\_t method), 125

get\_mangled\_name() (typedef\_t method), 136

get\_mangled\_name() (variable\_t method), 137

get\_members() (class\_t method), 72

get\_name2value\_dict() (enumeration\_t method), 107

get\_named\_parent() (in module pygccxml.declarations.declaration\_utils), 104

get\_os\_file\_names() (project\_reader\_t static method), 150

get\_partial\_name() (in module pygccxml.declarations.class\_declaration), 75

get\_single() (matcher static method), 122

get\_string\_repr() (xml\_generators method), 156

get\_tr1() (in module pygccxml.utils.utils), 155

getter() (cached method), 154

guess\_calling\_convention() (calldef\_t method), 52

guess\_calling\_convention() (casting\_operator\_t method), 54

guess\_calling\_convention() (constructor\_t method), 56

guess\_calling\_convention() (destructor\_t method), 58

guess\_calling\_convention() (free\_calldef\_t method), 109

guess\_calling\_convention() (free\_function\_t method), 110

guess\_calling\_convention() (free\_operator\_t method), 112

guess\_calling\_convention() (member\_calldef\_t method), 60

guess\_calling\_convention() (member\_function\_t method), 62

guess\_calling\_convention() (member\_operator\_t method), 64

guess\_calling\_convention() (operator\_t method), 66

## H

has\_any\_non\_copyconstructor() (in module pygccxml.declarations.type\_traits\_classes), 132

has\_const (casting\_operator\_t attribute), 54

has\_const (constructor\_t attribute), 56

has\_const (destructor\_t attribute), 58

has\_const (member\_calldef\_t attribute), 60

has\_const (member\_function\_t attribute), 62

has\_const (member\_function\_type\_t attribute), 93

has\_const (member\_operator\_t attribute), 64

has\_copy\_constructor() (in module pygccxml.declarations.type\_traits\_classes), 132

has\_destructor() (in module pygccxml.declarations.type\_traits\_classes), 132

has\_ellipsis (calldef\_t attribute), 52

has\_ellipsis (calldef\_type\_t attribute), 78

has\_ellipsis (casting\_operator\_t attribute), 54

has\_ellipsis (constructor\_t attribute), 56

has\_ellipsis (destructor\_t attribute), 58

has\_ellipsis (free\_calldef\_t attribute), 109

has\_ellipsis (free\_function\_t attribute), 111

has\_ellipsis (free\_function\_type\_t attribute), 84

has\_ellipsis (free\_operator\_t attribute), 112

has\_ellipsis (member\_calldef\_t attribute), 60

has\_ellipsis (member\_function\_t attribute), 62

has\_ellipsis (member\_function\_type\_t attribute), 93

has\_ellipsis (member\_operator\_t attribute), 64

has\_ellipsis (operator\_t attribute), 66

has\_extern (calldef\_t attribute), 52

has\_extern (casting\_operator\_t attribute), 54

[has\\_extern \(constructor\\_t attribute\), 56](#)  
[has\\_extern \(destructor\\_t attribute\), 58](#)  
[has\\_extern \(free\\_calldef\\_t attribute\), 109](#)  
[has\\_extern \(free\\_function\\_t attribute\), 111](#)  
[has\\_extern \(free\\_operator\\_t attribute\), 112](#)  
[has\\_extern \(member\\_calldef\\_t attribute\), 60](#)  
[has\\_extern \(member\\_function\\_t attribute\), 62](#)  
[has\\_extern \(member\\_operator\\_t attribute\), 64](#)  
[has\\_extern \(operator\\_t attribute\), 66](#)  
[has\\_extern \(type\\_qualifiers\\_t attribute\), 96](#)  
[has\\_inline \(calldef\\_t attribute\), 52](#)  
[has\\_inline \(casting\\_operator\\_t attribute\), 54](#)  
[has\\_inline \(constructor\\_t attribute\), 56](#)  
[has\\_inline \(destructor\\_t attribute\), 58](#)  
[has\\_inline \(free\\_calldef\\_t attribute\), 109](#)  
[has\\_inline \(free\\_function\\_t attribute\), 111](#)  
[has\\_inline \(free\\_operator\\_t attribute\), 113](#)  
[has\\_inline \(member\\_calldef\\_t attribute\), 60](#)  
[has\\_inline \(member\\_function\\_t attribute\), 62](#)  
[has\\_inline \(member\\_operator\\_t attribute\), 64](#)  
[has\\_inline \(operator\\_t attribute\), 66](#)  
[has\\_mutable \(type\\_qualifiers\\_t attribute\), 96](#)  
[has\\_pattern\(\) \(parser\\_t method\), 120](#)  
[has\\_public\\_assign\(\) \(in module pygccxml.declarations.type\\_traits\\_classes\), 132](#)  
[has\\_public\\_binary\\_operator\(\) \(in module pygccxml.declarations.has\\_operator\\_matcher\), 114](#)  
[has\\_public\\_constructor\(\) \(in module pygccxml.declarations.type\\_traits\\_classes\), 132](#)  
[has\\_public\\_destructor\(\) \(in module pygccxml.declarations.type\\_traits\\_classes\), 132](#)  
[has\\_public\\_equal\(\) \(in module pygccxml.declarations.has\\_operator\\_matcher\), 114](#)  
[has\\_public\\_less\(\) \(in module pygccxml.declarations.has\\_operator\\_matcher\), 114](#)  
[has\\_static \(casting\\_operator\\_t attribute\), 54](#)  
[has\\_static \(constructor\\_t attribute\), 56](#)  
[has\\_static \(destructor\\_t attribute\), 58](#)  
[has\\_static \(member\\_calldef\\_t attribute\), 60](#)  
[has\\_static \(member\\_function\\_t attribute\), 62](#)  
[has\\_static \(member\\_operator\\_t attribute\), 64](#)  
[has\\_static \(type\\_qualifiers\\_t attribute\), 96](#)  
[has\\_trivial\\_constructor\(\) \(in module pygccxml.declarations.type\\_traits\\_classes\), 132](#)  
[has\\_value\\_name\(\) \(enumeration\\_t method\), 107](#)  
[has\\_vtable\(\) \(in module pygccxml.declarations.type\\_traits\\_classes\), 132](#)  
[hierarchy\\_info\\_t \(class in pygccxml.declarations.class\\_declaration\), 75](#)  
[hint \(dependency\\_info\\_t attribute\), 75](#)  
  
[i\\_depend\\_on\\_them\(\) \(calldef\\_t method\), 52](#)  
[i\\_depend\\_on\\_them\(\) \(casting\\_operator\\_t method\), 54](#)  
[i\\_depend\\_on\\_them\(\) \(class\\_declaration\\_t method\), 69](#)  
[i\\_depend\\_on\\_them\(\) \(class\\_t method\), 72](#)  
[i\\_depend\\_on\\_them\(\) \(constructor\\_t method\), 56](#)  
[i\\_depend\\_on\\_them\(\) \(declaration\\_t method\), 103](#)  
[i\\_depend\\_on\\_them\(\) \(dependency\\_info\\_t static method\), 75](#)  
[i\\_depend\\_on\\_them\(\) \(destructor\\_t method\), 58](#)  
[i\\_depend\\_on\\_them\(\) \(enumeration\\_t method\), 107](#)  
[i\\_depend\\_on\\_them\(\) \(free\\_calldef\\_t method\), 109](#)  
[i\\_depend\\_on\\_them\(\) \(free\\_function\\_t method\), 111](#)  
[i\\_depend\\_on\\_them\(\) \(free\\_operator\\_t method\), 113](#)  
[i\\_depend\\_on\\_them\(\) \(member\\_calldef\\_t method\), 60](#)  
[i\\_depend\\_on\\_them\(\) \(member\\_function\\_t method\), 62](#)  
[i\\_depend\\_on\\_them\(\) \(member\\_operator\\_t method\), 65](#)  
[i\\_depend\\_on\\_them\(\) \(namespace\\_t method\), 118](#)  
[i\\_depend\\_on\\_them\(\) \(operator\\_t method\), 66](#)  
[i\\_depend\\_on\\_them\(\) \(scopedef\\_t method\), 125](#)  
[i\\_depend\\_on\\_them\(\) \(typedef\\_t method\), 136](#)  
[i\\_depend\\_on\\_them\(\) \(variable\\_t method\), 137](#)  
[ietree\\_scanner\\_t \(class in pygccxml.parser.etree\\_scanner\), 144](#)  
[ignorableWhitespace\(\) \(ietree\\_scanner\\_t method\), 144](#)  
[ignorableWhitespace\(\) \(scanner\\_t method\), 151](#)  
[ignore\\_gccxml\\_output \(xml\\_generator\\_configuration\\_t attribute\), 140](#)  
[impl\\_details \(class in pygccxml.declarations.class\\_declaration\), 75](#)  
[impl\\_details \(class in pygccxml.declarations.traits\\_impl\\_details\), 129](#)  
[inc\\_ref\\_count\(\) \(filename\\_entry\\_t method\), 143](#)  
[include\\_paths \(parser\\_configuration\\_t attribute\), 139](#)  
[include\\_paths \(xml\\_generator\\_configuration\\_t attribute\), 140](#)  
[included\\_files \(record\\_t attribute\), 142](#)  
[included\\_files\\_signature \(record\\_t attribute\), 142](#)  
[INDENT\\_SIZE \(decl\\_printer\\_t attribute\), 101](#)  
[index\\_entry\\_t \(class in pygccxml.parser.directory\\_cache\), 144](#)  
[init\\_optimizer\(\) \(class\\_t method\), 72](#)  
[init\\_optimizer\(\) \(namespace\\_t method\), 118](#)  
[init\\_optimizer\(\) \(scopedef\\_t method\), 125](#)  
[instance \(decl\\_printer\\_t attribute\), 101](#)  
[instance \(linker\\_t attribute\), 146](#)  
[int128\\_t \(class in pygccxml.declarations.cpptypes\), 85](#)  
[int\\_t \(class in pygccxml.declarations.cpptypes\), 85](#)  
[internal\\_type\\_traits \(class in pygccxml.declarations.smart\\_pointer\\_traits\), 127](#)  
[is\\_abstract \(class\\_t attribute\), 73](#)  
[is\\_arithmetic\(\) \(in module pygccxml.declarations.type\\_traits\), 129](#)



`is_array()` (in module `pygccxml.declarations.type_traits`), 129

`is_artificial` (`calldef_t` attribute), 52

`is_artificial` (`casting_operator_t` attribute), 54

`is_artificial` (`class_declaration_t` attribute), 69

`is_artificial` (`class_t` attribute), 73

`is_artificial` (`constructor_t` attribute), 56

`is_artificial` (`declaration_t` attribute), 103

`is_artificial` (`destructor_t` attribute), 58

`is_artificial` (`enumeration_t` attribute), 107

`is_artificial` (`free_calldef_t` attribute), 109

`is_artificial` (`free_function_t` attribute), 111

`is_artificial` (`free_operator_t` attribute), 113

`is_artificial` (`member_calldef_t` attribute), 60

`is_artificial` (`member_function_t` attribute), 62

`is_artificial` (`member_operator_t` attribute), 65

`is_artificial` (`namespace_t` attribute), 118

`is_artificial` (`operator_t` attribute), 67

`is_artificial` (`scopedef_t` attribute), 125

`is_artificial` (`typedef_t` attribute), 136

`is_artificial` (`variable_t` attribute), 137

`is_base_and_derived()` (in module `pygccxml.declarations.type_traits_classes`), 132

`is_binary_operator()` (in module `pygccxml.declarations.type_traits_classes`), 132

`is_bool()` (in module `pygccxml.declarations.type_traits`), 129

`is_builtin_decl()` (`decl_printer_t` static method), 101

`is_call_invocation()` (in module `pygccxml.declarations.call_invocation`), 50

`is_calldef_pointer()` (in module `pygccxml.declarations.type_traits`), 129

`is_castxml` (`xml_generators` attribute), 156

`is_castxml1` (`xml_generators` attribute), 156

`is_class` (in module `pygccxml.declarations.type_traits_classes`), 132

`is_class_declaration` (in module `pygccxml.declarations.type_traits_classes`), 133

`is_const()` (in module `pygccxml.declarations.type_traits`), 129

`is_convertible()` (in module `pygccxml.declarations.type_traits_classes`), 133

`is_copy_constructor()` (in module `pygccxml.declarations.type_traits_classes`), 133

`is_cxx03` (`cxx_standard` attribute), 154

`is_cxx11` (`cxx_standard` attribute), 154

`is_cxx11_or_greater` (`cxx_standard` attribute), 154

`is_cxx14` (`cxx_standard` attribute), 154

`is_cxx14_or_greater` (`cxx_standard` attribute), 154

`is_cxx1z` (`cxx_standard` attribute), 155

`is_defined_in_xxx()` (`impl_details` static method), 129

`is_elaborated()` (in module `pygccxml.declarations.type_traits`), 130

`is_enum` (in module `pygccxml.declarations.type_traits_classes`), 133

`is_file_modified()` (`filename_repository_t` method), 143

`is_floating_point()` (in module `pygccxml.declarations.type_traits`), 130

`is_full_name()` (`calldef_matcher_t` method), 105

`is_full_name()` (`declaration_matcher_t` method), 105

`is_full_name()` (`namespace_matcher_t` method), 105

`is_full_name()` (`operator_matcher_t` method), 105

`is_full_name()` (`variable_matcher_t` method), 105

`is_fundamental()` (in module `pygccxml.declarations.type_traits`), 130

`is_gccxml` (`xml_generators` attribute), 156

`is_gccxml_06` (`xml_generators` attribute), 156

`is_gccxml_07` (`xml_generators` attribute), 156

`is_gccxml_09` (`xml_generators` attribute), 157

`is_gccxml_09_buggy` (`xml_generators` attribute), 157

`is_implicit` (`cxx_standard` attribute), 155

`is_instantiation()` (in module `pygccxml.declarations.templates`), 128

`is_integral()` (in module `pygccxml.declarations.type_traits`), 130

`is_mapping()` (`container_traits_impl_t` method), 76

`is_my_case()` (`container_traits_impl_t` method), 76

`is_my_case()` (`declaration_xxx_traits` method), 131

`is_noncopyable()` (in module `pygccxml.declarations.type_traits_classes`), 133

`is_pointer()` (in module `pygccxml.declarations.type_traits`), 130

`is_reference()` (in module `pygccxml.declarations.type_traits`), 130

`is_same()` (in module `pygccxml.declarations.type_traits`), 130

`is_same_function()` (in module `pygccxml.declarations.function_traits`), 114

`is_same_return_type()` (in module `pygccxml.declarations.function_traits`), 114

`is_sequence()` (`container_traits_impl_t` method), 76

`is_smart_pointer()` (`auto_ptr_traits` static method), 127

`is_smart_pointer()` (`smart_pointer_traits` static method), 127

`is_std_ostream()` (in module `pygccxml.declarations.type_traits`), 130

`is_std_string()` (in module `pygccxml.declarations.type_traits`), 130

`is_std_wostream()` (in module `pygccxml.declarations.type_traits`), 130

`is_std_wstring()` (in module `pygccxml.declarations.type_traits`), 130

`is_str()` (in module `pygccxml.utils.utils`), 155

`is_struct()` (in module `pygccxml.declarations.type_traits_classes`), 133

`is_trivial_constructor()` (in module `pygccxml.declarations.type_traits_classes`), 133

- is\_unary\_operator() (in module pygccxml.declarations.type\_traits\_classes), 133
  - is\_union() (in module pygccxml.declarations.type\_traits\_classes), 133
  - is\_virtual (hierarchy\_info\_t attribute), 75
  - is\_void() (in module pygccxml.declarations.type\_traits), 130
  - is\_void\_pointer() (in module pygccxml.declarations.type\_traits), 130
  - is\_volatile() (in module pygccxml.declarations.type\_traits), 130
- ## J
- java\_fundamental\_t (class in pygccxml.declarations.cpptypes), 86
  - jboolean\_t (class in pygccxml.declarations.cpptypes), 86
  - jbyte\_t (class in pygccxml.declarations.cpptypes), 87
  - jchar\_t (class in pygccxml.declarations.cpptypes), 87
  - jdouble\_t (class in pygccxml.declarations.cpptypes), 88
  - jfloat\_t (class in pygccxml.declarations.cpptypes), 88
  - jint\_t (class in pygccxml.declarations.cpptypes), 88
  - jlong\_t (class in pygccxml.declarations.cpptypes), 89
  - JNAME (jboolean\_t attribute), 86
  - JNAME (jbyte\_t attribute), 87
  - JNAME (jchar\_t attribute), 87
  - JNAME (jdouble\_t attribute), 88
  - JNAME (jfloat\_t attribute), 88
  - JNAME (jint\_t attribute), 89
  - JNAME (jlong\_t attribute), 89
  - JNAME (jshort\_t attribute), 89
  - join() (in module pygccxml.declarations.call\_invocation), 50
  - join() (in module pygccxml.declarations.templates), 128
  - join() (parser\_t method), 121
  - join\_declarations() (in module pygccxml.parser.declarations\_joiner), 142
  - join\_declarations() (source\_reader\_t method), 153
  - jshort\_t (class in pygccxml.declarations.cpptypes), 89
  - JUSTIFY (decl\_printer\_t attribute), 101
- ## K
- keep\_xml (parser\_configuration\_t attribute), 139
  - keep\_xml (xml\_generator\_configuration\_t attribute), 140
  - key() (record\_t method), 142
  - key\_type() (container\_traits\_impl\_t method), 76
- ## L
- level (decl\_printer\_t attribute), 101
  - line (location\_t attribute), 114
  - linker\_t (class in pygccxml.parser.linker), 146
  - load\_xml\_generator\_configuration() (in module pygccxml.parser.config), 138
  - location (calldef\_t attribute), 52
  - location (casting\_operator\_t attribute), 54
  - location (class\_declaration\_t attribute), 69
  - location (class\_t attribute), 73
  - location (constructor\_t attribute), 57
  - location (declaration\_t attribute), 103
  - location (destructor\_t attribute), 59
  - location (enumeration\_t attribute), 107
  - location (free\_calldef\_t attribute), 109
  - location (free\_function\_t attribute), 111
  - location (free\_operator\_t attribute), 113
  - location (member\_calldef\_t attribute), 61
  - location (member\_function\_t attribute), 63
  - location (member\_operator\_t attribute), 65
  - location (namespace\_t attribute), 118
  - location (operator\_t attribute), 67
  - location (scopedef\_t attribute), 125
  - location (typedef\_t attribute), 136
  - location (variable\_t attribute), 137
  - location\_t (class in pygccxml.declarations.location), 114
  - logger (cache\_base\_t attribute), 141
  - logger (directory\_cache\_t attribute), 143
  - logger (dummy\_cache\_t attribute), 141
  - logger (file\_cache\_t attribute), 142
  - loggers (class in pygccxml.utils.utils), 155
  - long\_double\_t (class in pygccxml.declarations.cpptypes), 90
  - long\_int\_t (class in pygccxml.declarations.cpptypes), 90
  - long\_long\_int\_t (class in pygccxml.declarations.cpptypes), 91
  - long\_long\_unsigned\_int\_t (class in pygccxml.declarations.cpptypes), 91
  - long\_unsigned\_int\_t (class in pygccxml.declarations.cpptypes), 92
- ## M
- make\_flatten() (in module pygccxml.declarations.scopedef), 122
  - mangled (calldef\_t attribute), 52
  - mangled (casting\_operator\_t attribute), 54
  - mangled (class\_declaration\_t attribute), 69
  - mangled (class\_t attribute), 73
  - mangled (constructor\_t attribute), 57
  - mangled (declaration\_t attribute), 103
  - mangled (destructor\_t attribute), 59
  - mangled (enumeration\_t attribute), 107
  - mangled (free\_calldef\_t attribute), 109
  - mangled (free\_function\_t attribute), 111
  - mangled (free\_operator\_t attribute), 113
  - mangled (member\_calldef\_t attribute), 61
  - mangled (member\_function\_t attribute), 63
  - mangled (member\_operator\_t attribute), 65
  - mangled (namespace\_t attribute), 118
  - mangled (operator\_t attribute), 67
  - mangled (scopedef\_t attribute), 125
  - mangled (typedef\_t attribute), 136

[mangled \(variable\\_t attribute\), 137](#)  
[match\\_declaration\\_t \(class in pygccxml.declarations.algorithm\), 48](#)  
[matcher \(class in pygccxml.declarations.scopedef\), 122](#)  
[matcher\\_base\\_t \(class in pygccxml.declarations.matchers\), 115](#)  
[mdecl\\_wrapper\\_t \(class in pygccxml.declarations.mdecl\\_wrapper\), 115](#)  
[mem\\_fun\(\) \(class\\_t method\), 73](#)  
[mem\\_fun\(\) \(namespace\\_t method\), 118](#)  
[mem\\_fun\(\) \(scopedef\\_t method\), 125](#)  
[mem\\_funs\(\) \(class\\_t method\), 73](#)  
[mem\\_funs\(\) \(namespace\\_t method\), 119](#)  
[mem\\_funs\(\) \(scopedef\\_t method\), 125](#)  
[mem\\_oper\(\) \(class\\_t method\), 73](#)  
[mem\\_oper\(\) \(namespace\\_t method\), 119](#)  
[mem\\_oper\(\) \(scopedef\\_t method\), 126](#)  
[mem\\_ops\(\) \(class\\_t method\), 73](#)  
[mem\\_ops\(\) \(namespace\\_t method\), 119](#)  
[mem\\_ops\(\) \(scopedef\\_t method\), 126](#)  
[member\\_calldf\\_t \(class in pygccxml.declarations.calldf\\_members\), 59](#)  
[member\\_function\(\) \(class\\_t method\), 73](#)  
[member\\_function\(\) \(namespace\\_t method\), 119](#)  
[member\\_function\(\) \(scopedef\\_t method\), 126](#)  
[member\\_function\\_t \(class in pygccxml.declarations.calldf\\_members\), 61](#)  
[member\\_function\\_type\\_t \(class in pygccxml.declarations.cpptypes\), 92](#)  
[member\\_functions\(\) \(class\\_t method\), 73](#)  
[member\\_functions\(\) \(namespace\\_t method\), 119](#)  
[member\\_functions\(\) \(scopedef\\_t method\), 126](#)  
[member\\_operator\(\) \(class\\_t method\), 73](#)  
[member\\_operator\(\) \(namespace\\_t method\), 119](#)  
[member\\_operator\(\) \(scopedef\\_t method\), 126](#)  
[member\\_operator\\_t \(class in pygccxml.declarations.calldf\\_members\), 63](#)  
[member\\_operators\(\) \(class\\_t method\), 73](#)  
[member\\_operators\(\) \(namespace\\_t method\), 119](#)  
[member\\_operators\(\) \(scopedef\\_t method\), 126](#)  
[member\\_variable\\_type\\_t \(class in pygccxml.declarations.cpptypes\), 93](#)  
[members\(\) \(ietree\\_scanner\\_t method\), 145](#)  
[members\(\) \(scanner\\_t method\), 151](#)  
[message \(declaration\\_not\\_found\\_t attribute\), 121](#)  
[message \(multiple\\_declarations\\_found\\_t attribute\), 121](#)  
[message \(visit\\_function\\_has\\_not\\_been\\_found\\_t attribute\), 121](#)  
[multiple\\_declarations\\_found\\_t, 121](#)  
[multiple\\_declarations\\_found\\_t \(class\\_t attribute\), 73](#)  
[multiple\\_declarations\\_found\\_t \(namespace\\_t attribute\), 119](#)  
[multiple\\_declarations\\_found\\_t \(scopedef\\_t attribute\), 126](#)

## N

[name \(argument\\_t attribute\), 51](#)  
[name \(calldf\\_matcher\\_t attribute\), 105](#)  
[name \(calldf\\_t attribute\), 53](#)  
[name \(casting\\_operator\\_t attribute\), 55](#)  
[name \(class\\_declaration\\_t attribute\), 69](#)  
[name \(class\\_t attribute\), 74](#)  
[name \(constructor\\_t attribute\), 57](#)  
[name \(declaration\\_matcher\\_t attribute\), 105](#)  
[name \(declaration\\_t attribute\), 103](#)  
[name \(destructor\\_t attribute\), 59](#)  
[name \(enumeration\\_t attribute\), 107](#)  
[name \(free\\_calldf\\_t attribute\), 109](#)  
[name \(free\\_function\\_t attribute\), 111](#)  
[name \(free\\_operator\\_t attribute\), 113](#)  
[name \(member\\_calldf\\_t attribute\), 61](#)  
[name \(member\\_function\\_t attribute\), 63](#)  
[name \(member\\_operator\\_t attribute\), 65](#)  
[name \(namespace\\_matcher\\_t attribute\), 105](#)  
[name \(namespace\\_t attribute\), 119](#)  
[name \(operator\\_matcher\\_t attribute\), 105](#)  
[name \(operator\\_t attribute\), 67](#)  
[name \(scopedef\\_t attribute\), 126](#)  
[name \(typedef\\_t attribute\), 136](#)  
[name \(variable\\_matcher\\_t attribute\), 106](#)  
[name \(variable\\_t attribute\), 137](#)  
[name\(\) \(container\\_traits\\_impl\\_t method\), 76](#)  
[name\(\) \(in module pygccxml.declarations.call\\_invocation\), 50](#)  
[name\(\) \(in module pygccxml.declarations.templates\), 128](#)  
[name\(\) \(parser\\_t method\), 121](#)  
[NAME\\_TEMPLATE \(free\\_function\\_type\\_t attribute\), 84](#)  
[NAME\\_TEMPLATE \(member\\_function\\_type\\_t attribute\), 92](#)  
[NAME\\_TEMPLATE \(member\\_variable\\_type\\_t attribute\), 93](#)  
[namespace\(\) \(namespace\\_t method\), 119](#)  
[namespace\\_matcher \(in module pygccxml.declarations\), 48](#)  
[namespace\\_matcher\\_t \(class in pygccxml.declarations.declarations\\_matchers\), 105](#)  
[namespace\\_t \(class in pygccxml.declarations.namespace\), 116](#)  
[namespaces\(\) \(namespace\\_t method\), 119](#)  
[no\\_const\(\) \(defaults\\_eraser method\), 77](#)  
[no\\_end\\_const\(\) \(defaults\\_eraser method\), 77](#)  
[no\\_gnustd\(\) \(defaults\\_eraser method\), 77](#)  
[no\\_std\(\) \(defaults\\_eraser method\), 77](#)  
[no\\_stdext\(\) \(defaults\\_eraser method\), 77](#)  
[normalize\(\) \(defaults\\_eraser method\), 77](#)  
[normalize\(\) \(in module pygccxml.declarations.templates\), 128](#)  
[normalize\(\) \(parser\\_t method\), 121](#)



- normalize\_full\_name\_false() (in module pygccxml.declarations.templates), 128
- normalize\_full\_name\_true() (in module pygccxml.declarations.templates), 128
- normalize\_name() (in module pygccxml.declarations.templates), 128
- normalize\_partial\_name() (in module pygccxml.declarations.templates), 128
- normalize\_path() (in module pygccxml.utils.utils), 156
- normalized\_full\_name\_false (declaration\_algs\_cache\_t attribute), 49
- normalized\_full\_name\_true (declaration\_algs\_cache\_t attribute), 49
- normalized\_name (declaration\_algs\_cache\_t attribute), 49
- normalized\_partial\_name (declaration\_algs\_cache\_t attribute), 49
- NOT\_FOUND (parser\_t attribute), 120
- not\_matcher (in module pygccxml.declarations), 48
- not\_matcher\_t (class in pygccxml.declarations.matchers), 115
- NOT\_VIRTUAL (VIRTUALITY\_TYPES attribute), 68
- nss() (namespace\_t method), 119
- ## O
- on\_missing\_functionality() (in module pygccxml.declarations.xml\_generators), 138
- operator() (class\_t method), 74
- operator() (namespace\_t method), 119
- operator() (scopedef\_t method), 126
- operator\_matcher (in module pygccxml.declarations), 48
- operator\_matcher\_t (class in pygccxml.declarations.declarations\_matchers), 105
- operator\_t (class in pygccxml.declarations.calldef\_members), 65
- OPERATOR\_WORD\_LEN (casting\_operator\_t attribute), 53
- OPERATOR\_WORD\_LEN (free\_operator\_t attribute), 112
- OPERATOR\_WORD\_LEN (member\_operator\_t attribute), 63
- OPERATOR\_WORD\_LEN (operator\_t attribute), 66
- operators() (class\_t method), 74
- operators() (namespace\_t method), 119
- operators() (scopedef\_t method), 126
- optional\_args (calldef\_t attribute), 53
- optional\_args (casting\_operator\_t attribute), 55
- optional\_args (constructor\_t attribute), 57
- optional\_args (destructor\_t attribute), 59
- optional\_args (free\_calldef\_t attribute), 109
- optional\_args (free\_function\_t attribute), 111
- optional\_args (free\_operator\_t attribute), 113
- optional\_args (member\_calldef\_t attribute), 61
- optional\_args (member\_function\_t attribute), 63
- optional\_args (member\_operator\_t attribute), 65
- optional\_args (operator\_t attribute), 67
- or\_matcher (in module pygccxml.declarations), 48
- or\_matcher\_t (class in pygccxml.declarations.matchers), 115
- overloads (calldef\_t attribute), 53
- overloads (casting\_operator\_t attribute), 55
- overloads (constructor\_t attribute), 57
- overloads (destructor\_t attribute), 59
- overloads (free\_calldef\_t attribute), 109
- overloads (free\_function\_t attribute), 111
- overloads (free\_operator\_t attribute), 113
- overloads (member\_calldef\_t attribute), 61
- overloads (member\_function\_t attribute), 63
- overloads (member\_operator\_t attribute), 65
- overloads (operator\_t attribute), 67
- ## P
- parent (calldef\_t attribute), 53
- parent (casting\_operator\_t attribute), 55
- parent (class\_declaration\_t attribute), 69
- parent (class\_t attribute), 74
- parent (constructor\_t attribute), 57
- parent (declaration\_t attribute), 103
- parent (destructor\_t attribute), 59
- parent (enumeration\_t attribute), 107
- parent (free\_calldef\_t attribute), 109
- parent (free\_function\_t attribute), 111
- parent (free\_operator\_t attribute), 113
- parent (member\_calldef\_t attribute), 61
- parent (member\_function\_t attribute), 63
- parent (member\_operator\_t attribute), 65
- parent (namespace\_t attribute), 119
- parent (operator\_t attribute), 67
- parent (scopedef\_t attribute), 126
- parent (typedef\_t attribute), 136
- parent (variable\_t attribute), 137
- parse() (in module pygccxml.parser), 138
- parse\_string() (in module pygccxml.parser), 138
- parse\_xml\_file() (in module pygccxml.parser), 138
- parser\_configuration\_t (class in pygccxml.parser.config), 139
- parser\_t (class in pygccxml.declarations.pattern\_parser), 120
- partial\_decl\_string (array\_t attribute), 78
- partial\_decl\_string (bool\_t attribute), 78
- partial\_decl\_string (calldef\_t attribute), 53
- partial\_decl\_string (casting\_operator\_t attribute), 55
- partial\_decl\_string (char\_t attribute), 79
- partial\_decl\_string (class\_declaration\_t attribute), 70
- partial\_decl\_string (class\_t attribute), 74
- partial\_decl\_string (complex\_double\_t attribute), 79
- partial\_decl\_string (complex\_float\_t attribute), 80

`partial_decl_string` (`complex_long_double_t` attribute), 80

`partial_decl_string` (`compound_t` attribute), 81

`partial_decl_string` (`const_t` attribute), 81

`partial_decl_string` (`constructor_t` attribute), 57

`partial_decl_string` (`declared_t` attribute), 81

`partial_decl_string` (`declaration_t` attribute), 103

`partial_decl_string` (`destructor_t` attribute), 59

`partial_decl_string` (`double_t` attribute), 82

`partial_decl_string` (`dummy_type_t` attribute), 82

`partial_decl_string` (`elaborated_t` attribute), 83

`partial_decl_string` (`ellipsis_t` attribute), 83

`partial_decl_string` (`enumeration_t` attribute), 108

`partial_decl_string` (`float_t` attribute), 84

`partial_decl_string` (`free_calldef_t` attribute), 109

`partial_decl_string` (`free_function_t` attribute), 111

`partial_decl_string` (`free_function_type_t` attribute), 84

`partial_decl_string` (`free_operator_t` attribute), 113

`partial_decl_string` (`fundamental_t` attribute), 85

`partial_decl_string` (`int128_t` attribute), 85

`partial_decl_string` (`int_t` attribute), 86

`partial_decl_string` (`java_fundamental_t` attribute), 86

`partial_decl_string` (`jboolean_t` attribute), 87

`partial_decl_string` (`jbyte_t` attribute), 87

`partial_decl_string` (`jchar_t` attribute), 88

`partial_decl_string` (`jdouble_t` attribute), 88

`partial_decl_string` (`jfloat_t` attribute), 88

`partial_decl_string` (`jint_t` attribute), 89

`partial_decl_string` (`jlong_t` attribute), 89

`partial_decl_string` (`jshort_t` attribute), 90

`partial_decl_string` (`long_double_t` attribute), 90

`partial_decl_string` (`long_int_t` attribute), 91

`partial_decl_string` (`long_long_int_t` attribute), 91

`partial_decl_string` (`long_long_unsigned_int_t` attribute), 92

`partial_decl_string` (`long_unsigned_int_t` attribute), 92

`partial_decl_string` (`member_calldef_t` attribute), 61

`partial_decl_string` (`member_function_t` attribute), 63

`partial_decl_string` (`member_function_type_t` attribute), 93

`partial_decl_string` (`member_operator_t` attribute), 65

`partial_decl_string` (`member_variable_type_t` attribute), 93

`partial_decl_string` (`namespace_t` attribute), 119

`partial_decl_string` (`operator_t` attribute), 67

`partial_decl_string` (`pointer_t` attribute), 94

`partial_decl_string` (`reference_t` attribute), 94

`partial_decl_string` (`restrict_t` attribute), 95

`partial_decl_string` (`scopedef_t` attribute), 126

`partial_decl_string` (`short_int_t` attribute), 95

`partial_decl_string` (`short_unsigned_int_t` attribute), 96

`partial_decl_string` (`signed_char_t` attribute), 96

`partial_decl_string` (`type_algs_cache_t` attribute), 49

`partial_decl_string` (`type_t` attribute), 97

`partial_decl_string` (`typedef_t` attribute), 136

`partial_decl_string` (`uint128_t` attribute), 97

`partial_decl_string` (`unknown_t` attribute), 98

`partial_decl_string` (`unsigned_char_t` attribute), 98

`partial_decl_string` (`unsigned_int_t` attribute), 98

`partial_decl_string` (`variable_t` attribute), 137

`partial_decl_string` (`void_t` attribute), 99

`partial_decl_string` (`volatile_t` attribute), 99

`partial_decl_string` (`wchar_t` attribute), 100

`partial_declaration_path` (`declaration_algs_cache_t` attribute), 49

`partial_declaration_path()` (in module `pygccxml.declarations.declaration_utils`), 104

`partial_name` (`calldef_t` attribute), 53

`partial_name` (`casting_operator_t` attribute), 55

`partial_name` (`class_declaration_t` attribute), 70

`partial_name` (`class_t` attribute), 74

`partial_name` (`constructor_t` attribute), 57

`partial_name` (`declaration_t` attribute), 103

`partial_name` (`destructor_t` attribute), 59

`partial_name` (`enumeration_t` attribute), 108

`partial_name` (`free_calldef_t` attribute), 109

`partial_name` (`free_function_t` attribute), 111

`partial_name` (`free_operator_t` attribute), 113

`partial_name` (`member_calldef_t` attribute), 61

`partial_name` (`member_function_t` attribute), 63

`partial_name` (`member_operator_t` attribute), 65

`partial_name` (`namespace_t` attribute), 120

`partial_name` (`operator_t` attribute), 67

`partial_name` (`scopedef_t` attribute), 126

`partial_name` (`typedef_t` attribute), 136

`partial_name` (`variable_t` attribute), 137

`pattern` (`CALLING_CONVENTION_TYPES` attribute), 68

`pdb_reader` (`loggers` attribute), 155

`pointer_t` (class in `pygccxml.declarations.cpptypes`), 93

`print_calldef_info()` (`decl_printer_t` method), 101

`print_decl_header()` (`decl_printer_t` method), 101

`print_declarations()` (in module `pygccxml.declarations.decl_printer`), 102

`print_details` (`decl_printer_t` attribute), 101

`PRIVATE` (`ACCESS_TYPES` attribute), 68

`private_members` (`class_t` attribute), 74

`processingInstruction()` (`ietree_scanner_t` method), 145

`processingInstruction()` (`scanner_t` method), 151

`project_reader_t` (class in `pygccxml.parser.project_reader`), 149

`PROTECTED` (`ACCESS_TYPES` attribute), 68

`protected_members` (`class_t` attribute), 74

`PUBLIC` (`ACCESS_TYPES` attribute), 68

`public_members` (`class_t` attribute), 74

`PURE_VIRTUAL` (`VIRTUALITY_TYPES` attribute), 68

`pygccxml` (module), 47

`pygccxml.declarations` (module), 47

[pygccxml.declarations.algorithm \(module\)](#), 48  
[pygccxml.declarations.algorithms\\_cache \(module\)](#), 49  
[pygccxml.declarations.byte\\_info \(module\)](#), 50  
[pygccxml.declarations.call\\_invocation \(module\)](#), 50  
[pygccxml.declarations.calldef \(module\)](#), 51  
[pygccxml.declarations.calldef\\_members \(module\)](#), 53  
[pygccxml.declarations.calldef\\_types \(module\)](#), 67  
[pygccxml.declarations.class\\_declaration \(module\)](#), 68  
[pygccxml.declarations.container\\_traits \(module\)](#), 75  
[pygccxml.declarations.cpptypes \(module\)](#), 77  
[pygccxml.declarations.decl\\_factory \(module\)](#), 100  
[pygccxml.declarations.decl\\_printer \(module\)](#), 101  
[pygccxml.declarations.decl\\_visitor \(module\)](#), 102  
[pygccxml.declarations.declaration \(module\)](#), 102  
[pygccxml.declarations.declaration\\_utils \(module\)](#), 104  
[pygccxml.declarations.declarations\\_matchers \(module\)](#), 104  
[pygccxml.declarations.elaborated\\_info \(module\)](#), 106  
[pygccxml.declarations.enumeration \(module\)](#), 106  
[pygccxml.declarations.free\\_calldef \(module\)](#), 108  
[pygccxml.declarations.function\\_traits \(module\)](#), 114  
[pygccxml.declarations.has\\_operator\\_matcher \(module\)](#), 114  
[pygccxml.declarations.location \(module\)](#), 114  
[pygccxml.declarations.matchers \(module\)](#), 114  
[pygccxml.declarations.mdecl\\_wrapper \(module\)](#), 115  
[pygccxml.declarations.namespace \(module\)](#), 116  
[pygccxml.declarations.pattern\\_parser \(module\)](#), 120  
[pygccxml.declarations.runtime\\_errors \(module\)](#), 121  
[pygccxml.declarations.scopeddef \(module\)](#), 121  
[pygccxml.declarations.smart\\_pointer\\_traits \(module\)](#), 127  
[pygccxml.declarations.templates \(module\)](#), 127  
[pygccxml.declarations.traits\\_impl\\_details \(module\)](#), 129  
[pygccxml.declarations.type\\_traits \(module\)](#), 129  
[pygccxml.declarations.type\\_traits\\_classes \(module\)](#), 131  
[pygccxml.declarations.type\\_visitor \(module\)](#), 134  
[pygccxml.declarations.typedef \(module\)](#), 135  
[pygccxml.declarations.variable \(module\)](#), 136  
[pygccxml.declarations.xml\\_generators \(module\)](#), 138  
[pygccxml.parser \(module\)](#), 138  
[pygccxml.parser.config \(module\)](#), 138  
[pygccxml.parser.declarations\\_cache \(module\)](#), 141  
[pygccxml.parser.declarations\\_joiner \(module\)](#), 142  
[pygccxml.parser.directory\\_cache \(module\)](#), 142  
[pygccxml.parser.etree\\_scanner \(module\)](#), 144  
[pygccxml.parser.linker \(module\)](#), 146  
[pygccxml.parser.patcher \(module\)](#), 148  
[pygccxml.parser.project\\_reader \(module\)](#), 148  
[pygccxml.parser.scanner \(module\)](#), 150  
[pygccxml.parser.source\\_reader \(module\)](#), 152  
[pygccxml.utils \(module\)](#), 153  
[pygccxml.utils.utils \(module\)](#), 154  
[pygccxml.utils.xml\\_generators \(module\)](#), 156

## Q

[queries\\_engine \(loggers attribute\)](#), 156

## R

[raise\\_on\\_wrong\\_settings\(\) \(parser\\_configuration\\_t method\)](#), 139  
[raise\\_on\\_wrong\\_settings\(\) \(xml\\_generator\\_configuration\\_t method\)](#), 140  
[read\(\) \(ietree\\_scanner\\_t method\)](#), 145  
[read\(\) \(scanner\\_t method\)](#), 151  
[read\\_cpp\\_source\\_file\(\) \(source\\_reader\\_t method\)](#), 153  
[read\\_file\(\) \(source\\_reader\\_t method\)](#), 153  
[read\\_files\(\) \(project\\_reader\\_t method\)](#), 150  
[read\\_string\(\) \(project\\_reader\\_t method\)](#), 150  
[read\\_string\(\) \(source\\_reader\\_t method\)](#), 153  
[read\\_xml\(\) \(project\\_reader\\_t method\)](#), 150  
[read\\_xml\\_file\(\) \(source\\_reader\\_t method\)](#), 153  
[record\\_t \(class in pygccxml.parser.declarations\\_cache\)](#), 142  
[recursive \(decl\\_printer\\_t attribute\)](#), 101  
[recursive\\_bases \(class\\_t attribute\)](#), 74  
[RECURSIVE\\_DEFAULT \(class\\_t attribute\)](#), 70  
[RECURSIVE\\_DEFAULT \(namespace\\_t attribute\)](#), 116  
[RECURSIVE\\_DEFAULT \(scopeddef\\_t attribute\)](#), 123  
[recursive\\_derived \(class\\_t attribute\)](#), 74  
[reference\\_t \(class in pygccxml.declarations.cpptypes\)](#), 94  
[regex\\_matcher \(in module pygccxml.declarations\)](#), 48  
[regex\\_matcher\\_t \(class in pygccxml.declarations.matchers\)](#), 115  
[related\\_class \(hierarchy\\_info\\_t attribute\)](#), 75  
[release\\_filename\(\) \(filename\\_repository\\_t method\)](#), 144  
[remove\\_alias \(type\\_algs\\_cache\\_t attribute\)](#), 49  
[remove\\_alias\(\) \(in module pygccxml.declarations.type\\_traits\)](#), 130  
[remove\\_const\(\) \(in module pygccxml.declarations.type\\_traits\)](#), 130  
[remove\\_cv\(\) \(in module pygccxml.declarations.type\\_traits\)](#), 131  
[remove\\_declarated\(\) \(in module pygccxml.declarations.type\\_traits\)](#), 131  
[remove\\_declaration\(\) \(class\\_t method\)](#), 74  
[remove\\_declaration\(\) \(namespace\\_t method\)](#), 120  
[remove\\_declaration\(\) \(scopeddef\\_t method\)](#), 126  
[remove\\_defaults\(\) \(container\\_traits\\_impl\\_t method\)](#), 76  
[remove\\_elaborated\(\) \(in module pygccxml.declarations.type\\_traits\)](#), 131  
[remove\\_file\\_no\\_raise\(\) \(in module pygccxml.utils.utils\)](#), 156  
[remove\\_pointer\(\) \(in module pygccxml.declarations.type\\_traits\)](#), 131  
[remove\\_reference\(\) \(in module pygccxml.declarations.type\\_traits\)](#), 131

`remove_volatile()` (in module `pygccxml.declarations.type_traits`), 131

`replace_basic_string()` (`defaults_eraser` method), 77

`required_args` (`calldef_t` attribute), 53

`required_args` (`casting_operator_t` attribute), 55

`required_args` (`constructor_t` attribute), 57

`required_args` (`destructor_t` attribute), 59

`required_args` (`free_calldef_t` attribute), 109

`required_args` (`free_function_t` attribute), 111

`required_args` (`free_operator_t` attribute), 113

`required_args` (`member_calldef_t` attribute), 61

`required_args` (`member_function_t` attribute), 63

`required_args` (`member_operator_t` attribute), 65

`required_args` (`operator_t` attribute), 67

`reset()` (`cached` method), 154

`reset()` (`declaration_algs_cache_t` method), 49

`reset()` (`type_algs_cache_t` method), 50

`reset_access_type()` (`declaration_algs_cache_t` method), 49

`reset_name_based()` (`declaration_algs_cache_t` method), 49

`restrict_t` (class in `pygccxml.declarations.cpptypes`), 94

`return_type` (`calldef_t` attribute), 53

`return_type` (`calldef_type_t` attribute), 78

`return_type` (`casting_operator_t` attribute), 55

`return_type` (`constructor_t` attribute), 57

`return_type` (`destructor_t` attribute), 59

`return_type` (`free_calldef_t` attribute), 110

`return_type` (`free_function_t` attribute), 111

`return_type` (`free_function_type_t` attribute), 85

`return_type` (`free_operator_t` attribute), 113

`return_type` (`member_calldef_t` attribute), 61

`return_type` (`member_function_t` attribute), 63

`return_type` (`member_function_type_t` attribute), 93

`return_type` (`member_operator_t` attribute), 65

`return_type` (`operator_t` attribute), 67

`root` (`loggers` attribute), 156

## S

`scanner_t` (class in `pygccxml.parser.scanner`), 150

`scopedef_t` (class in `pygccxml.declarations.scopedef`), 123

`sequential_container_traits` (in module `pygccxml.declarations.container_traits`), 77

`set_level()` (`loggers` static method), 156

`setDocumentLocator()` (`ietree_scanner_t` method), 145

`setDocumentLocator()` (`scanner_t` method), 151

`setter()` (`cached` method), 154

`short_int_t` (class in `pygccxml.declarations.cpptypes`), 95

`short_unsigned_int_t` (class in `pygccxml.declarations.cpptypes`), 95

`signed_char_t` (class in `pygccxml.declarations.cpptypes`), 96

`size` (`array_t` attribute), 78

`SIZE_UNKNOWN` (`array_t` attribute), 77

`skippedEntity()` (`ietree_scanner_t` method), 145

`skippedEntity()` (`scanner_t` method), 151

`smart_pointer_traits` (class in `pygccxml.declarations.smart_pointer_traits`), 127

`source_reader_t` (class in `pygccxml.parser.source_reader`), 152

`source_signature` (`record_t` attribute), 142

`split()` (in module `pygccxml.declarations.call_invocation`), 50

`split()` (in module `pygccxml.declarations.templates`), 128

`split()` (`parser_t` method), 121

`split_recursive()` (in module `pygccxml.declarations.call_invocation`), 51

`split_recursive()` (in module `pygccxml.declarations.templates`), 128

`split_recursive()` (`parser_t` method), 121

`STANDARD_SOURCE_FILE` (`file_configuration_t.CONTENT_TYPE` attribute), 149

`start_with_declarations` (`file_configuration_t` attribute), 149

`start_with_declarations` (`xml_generator_configuration_t` attribute), 140

`startDocument()` (`ietree_scanner_t` method), 145

`startDocument()` (`scanner_t` method), 151

`startElement()` (`ietree_scanner_t` method), 145

`startElement()` (`scanner_t` method), 152

`startElementNS()` (`ietree_scanner_t` method), 145

`startElementNS()` (`scanner_t` method), 152

`startPrefixMapping()` (`ietree_scanner_t` method), 145

`startPrefixMapping()` (`scanner_t` method), 152

`STDCALL` (`CALLING_CONVENTION_TYPES` attribute), 68

`stdcxx` (`cxx_standard` attribute), 155

`STRUCT` (`CLASS_TYPES` attribute), 69

`symbol` (`casting_operator_t` attribute), 55

`symbol` (`free_operator_t` attribute), 113

`symbol` (`member_operator_t` attribute), 65

`symbol` (`operator_t` attribute), 67

`SYSTEM_DEFAULT` (`CALLING_CONVENTION_TYPES` attribute), 68

## T

`take_parenting()` (`namespace_t` method), 120

`TEXT` (`file_configuration_t.CONTENT_TYPE` attribute), 149

`THISCALL` (`CALLING_CONVENTION_TYPES` attribute), 68

`to_list()` (`mdecl_wrapper_t` method), 116

`top_class` (`class_t` attribute), 74

`top_parent` (`calldef_t` attribute), 53

`top_parent` (`casting_operator_t` attribute), 55



[top\\_parent \(class\\_declaration\\_t attribute\), 70](#)  
[top\\_parent \(class\\_t attribute\), 74](#)  
[top\\_parent \(constructor\\_t attribute\), 57](#)  
[top\\_parent \(declaration\\_t attribute\), 103](#)  
[top\\_parent \(destructor\\_t attribute\), 59](#)  
[top\\_parent \(enumeration\\_t attribute\), 108](#)  
[top\\_parent \(free\\_calldef\\_t attribute\), 110](#)  
[top\\_parent \(free\\_function\\_t attribute\), 111](#)  
[top\\_parent \(free\\_operator\\_t attribute\), 113](#)  
[top\\_parent \(member\\_calldef\\_t attribute\), 61](#)  
[top\\_parent \(member\\_function\\_t attribute\), 63](#)  
[top\\_parent \(member\\_operator\\_t attribute\), 65](#)  
[top\\_parent \(namespace\\_t attribute\), 120](#)  
[top\\_parent \(operator\\_t attribute\), 67](#)  
[top\\_parent \(scopedef\\_t attribute\), 126](#)  
[top\\_parent \(typedef\\_t attribute\), 136](#)  
[top\\_parent \(variable\\_t attribute\), 138](#)  
[type\\_algs\\_cache\\_t \(class in pygccxml.declarations.algorithms\\_cache\), 49](#)  
[type\\_qualifiers \(variable\\_t attribute\), 138](#)  
[type\\_qualifiers\\_t \(class in pygccxml.declarations.cpptypes\), 96](#)  
[type\\_t \(class in pygccxml.declarations.cpptypes\), 96](#)  
[type\\_visitor\\_t \(class in pygccxml.declarations.type\\_visitor\), 134](#)  
[typedef\(\) \(class\\_t method\), 74](#)  
[typedef\(\) \(namespace\\_t method\), 120](#)  
[typedef\(\) \(scopedef\\_t method\), 126](#)  
[TYPEDEF\\_NAME\\_TEMPLATE \(free\\_function\\_type\\_t attribute\), 84](#)  
[TYPEDEF\\_NAME\\_TEMPLATE \(member\\_function\\_type\\_t attribute\), 92](#)  
[typedef\\_t \(class in pygccxml.declarations.typedef\), 135](#)  
[typedefs\(\) \(class\\_t method\), 74](#)  
[typedefs\(\) \(namespace\\_t method\), 120](#)  
[typedefs\(\) \(scopedef\\_t method\), 126](#)  
[types\(\) \(ietree\\_scanner\\_t method\), 146](#)  
[types\(\) \(scanner\\_t method\), 152](#)

## U

[uint128\\_t \(class in pygccxml.declarations.cpptypes\), 97](#)  
[undefine\\_symbols \(parser\\_configuration\\_t attribute\), 140](#)  
[undefine\\_symbols \(xml\\_generator\\_configuration\\_t attribute\), 140](#)  
[UNION \(CLASS\\_TYPES attribute\), 69](#)  
[UNKNOWN \(CALLING\\_CONVENTION\\_TYPES attribute\), 68](#)  
[unknown\\_t \(class in pygccxml.declarations.cpptypes\), 97](#)  
[unsigned\\_char\\_t \(class in pygccxml.declarations.cpptypes\), 98](#)  
[unsigned\\_int\\_t \(class in pygccxml.declarations.cpptypes\), 98](#)  
[update\(\) \(cache\\_base\\_t method\), 141](#)  
[update\(\) \(directory\\_cache\\_t method\), 143](#)

[update\(\) \(dummy\\_cache\\_t method\), 141](#)  
[update\(\) \(file\\_cache\\_t method\), 142](#)  
[update\\_id\\_counter\(\) \(filename\\_repository\\_t method\), 144](#)  
[update\\_unnamed\\_class\(\) \(in module pygccxml.parser.patcher\), 148](#)  
[USE\\_DEMANGLED\\_AS\\_NAME \(class\\_t attribute\), 70](#)  
[use\\_demangled\\_as\\_name \(class\\_t attribute\), 74](#)

## V

[value \(variable\\_t attribute\), 138](#)  
[value\\_type\(\) \(auto\\_ptr\\_traits static method\), 127](#)  
[value\\_type\(\) \(smart\\_pointer\\_traits static method\), 127](#)  
[values \(enumeration\\_t attribute\), 108](#)  
[variable\(\) \(class\\_t method\), 74](#)  
[variable\(\) \(namespace\\_t method\), 120](#)  
[variable\(\) \(scopedef\\_t method\), 127](#)  
[variable\\_matcher \(in module pygccxml.declarations\), 48](#)  
[variable\\_matcher\\_t \(class in pygccxml.declarations.declarations\\_matchers\), 105](#)  
[variable\\_t \(class in pygccxml.declarations.variable\), 136](#)  
[variable\\_type \(member\\_variable\\_type\\_t attribute\), 93](#)  
[variables\(\) \(class\\_t method\), 75](#)  
[variables\(\) \(namespace\\_t method\), 120](#)  
[variables\(\) \(scopedef\\_t method\), 127](#)  
[verbose \(decl\\_printer\\_t attribute\), 101](#)  
[VIRTUAL \(VIRTUALITY\\_TYPES attribute\), 68](#)  
[virtuality \(casting\\_operator\\_t attribute\), 55](#)  
[virtuality \(constructor\\_t attribute\), 57](#)  
[virtuality \(destructor\\_t attribute\), 59](#)  
[virtuality \(member\\_calldef\\_t attribute\), 61](#)  
[virtuality \(member\\_function\\_t attribute\), 63](#)  
[virtuality \(member\\_operator\\_t attribute\), 65](#)  
[virtuality\\_type\\_matcher \(in module pygccxml.declarations\), 48](#)  
[virtuality\\_type\\_matcher\\_t \(class in pygccxml.declarations.matchers\), 115](#)  
[VIRTUALITY\\_TYPES \(class in pygccxml.declarations.calldef\\_types\), 68](#)  
[visit\\_array\(\) \(linker\\_t method\), 146](#)  
[visit\\_array\(\) \(type\\_visitor\\_t method\), 134](#)  
[visit\\_bool\(\) \(linker\\_t method\), 146](#)  
[visit\\_bool\(\) \(type\\_visitor\\_t method\), 134](#)  
[visit\\_casting\\_operator\(\) \(decl\\_printer\\_t method\), 101](#)  
[visit\\_casting\\_operator\(\) \(decl\\_visitor\\_t method\), 102](#)  
[visit\\_casting\\_operator\(\) \(linker\\_t method\), 146](#)  
[visit\\_char\(\) \(linker\\_t method\), 146](#)  
[visit\\_char\(\) \(type\\_visitor\\_t method\), 134](#)  
[visit\\_class\(\) \(decl\\_printer\\_t method\), 101](#)  
[visit\\_class\(\) \(decl\\_visitor\\_t method\), 102](#)  
[visit\\_class\(\) \(linker\\_t method\), 146](#)  
[visit\\_class\\_declaration\(\) \(decl\\_printer\\_t method\), 101](#)  
[visit\\_class\\_declaration\(\) \(decl\\_visitor\\_t method\), 102](#)  
[visit\\_class\\_declaration\(\) \(linker\\_t method\), 146](#)

`visit_complex_double()` (linker\_t method), 146  
`visit_complex_double()` (type\_visitor\_t method), 134  
`visit_complex_float()` (linker\_t method), 146  
`visit_complex_float()` (type\_visitor\_t method), 134  
`visit_complex_long_double()` (linker\_t method), 146  
`visit_complex_long_double()` (type\_visitor\_t method), 134  
`visit_const()` (linker\_t method), 146  
`visit_const()` (type\_visitor\_t method), 134  
`visit_constructor()` (decl\_printer\_t method), 101  
`visit_constructor()` (decl\_visitor\_t method), 102  
`visit_constructor()` (linker\_t method), 146  
`visit_declarated()` (linker\_t method), 146  
`visit_declarated()` (type\_visitor\_t method), 134  
`visit_destructor()` (decl\_printer\_t method), 101  
`visit_destructor()` (decl\_visitor\_t method), 102  
`visit_destructor()` (linker\_t method), 146  
`visit_double()` (linker\_t method), 146  
`visit_double()` (type\_visitor\_t method), 134  
`visit_elaborated()` (linker\_t method), 146  
`visit_elaborated()` (type\_visitor\_t method), 134  
`visit_ellipsis()` (linker\_t method), 146  
`visit_ellipsis()` (type\_visitor\_t method), 134  
`visit_enumeration()` (decl\_printer\_t method), 101  
`visit_enumeration()` (decl\_visitor\_t method), 102  
`visit_enumeration()` (linker\_t method), 146  
`visit_float()` (linker\_t method), 146  
`visit_float()` (type\_visitor\_t method), 134  
`visit_free_function()` (decl\_printer\_t method), 101  
`visit_free_function()` (decl\_visitor\_t method), 102  
`visit_free_function()` (linker\_t method), 146  
`visit_free_function_type()` (linker\_t method), 146  
`visit_free_function_type()` (type\_visitor\_t method), 134  
`visit_free_operator()` (decl\_printer\_t method), 101  
`visit_free_operator()` (decl\_visitor\_t method), 102  
`visit_free_operator()` (linker\_t method), 147  
`visit_function_has_not_been_found_t`, 121  
`visit_int()` (linker\_t method), 147  
`visit_int()` (type\_visitor\_t method), 134  
`visit_int128()` (linker\_t method), 147  
`visit_int128()` (type\_visitor\_t method), 134  
`visit_jboolean()` (linker\_t method), 147  
`visit_jboolean()` (type\_visitor\_t method), 134  
`visit_jbyte()` (linker\_t method), 147  
`visit_jbyte()` (type\_visitor\_t method), 134  
`visit_jchar()` (linker\_t method), 147  
`visit_jchar()` (type\_visitor\_t method), 134  
`visit_jdouble()` (linker\_t method), 147  
`visit_jdouble()` (type\_visitor\_t method), 134  
`visit_jfloat()` (linker\_t method), 147  
`visit_jfloat()` (type\_visitor\_t method), 134  
`visit_jint()` (linker\_t method), 147  
`visit_jint()` (type\_visitor\_t method), 134  
`visit_jlong()` (linker\_t method), 147  
`visit_jlong()` (type\_visitor\_t method), 134  
`visit_jshort()` (linker\_t method), 147  
`visit_jshort()` (type\_visitor\_t method), 134  
`visit_long_double()` (linker\_t method), 147  
`visit_long_double()` (type\_visitor\_t method), 134  
`visit_long_int()` (linker\_t method), 147  
`visit_long_int()` (type\_visitor\_t method), 134  
`visit_long_long_int()` (linker\_t method), 147  
`visit_long_long_int()` (type\_visitor\_t method), 134  
`visit_long_long_unsigned_int()` (linker\_t method), 147  
`visit_long_long_unsigned_int()` (type\_visitor\_t method), 134  
`visit_long_unsigned_int()` (linker\_t method), 147  
`visit_long_unsigned_int()` (type\_visitor\_t method), 134  
`visit_member_function()` (decl\_printer\_t method), 101  
`visit_member_function()` (decl\_visitor\_t method), 102  
`visit_member_function()` (linker\_t method), 147  
`visit_member_function_type()` (linker\_t method), 147  
`visit_member_function_type()` (type\_visitor\_t method), 134  
`visit_member_operator()` (decl\_printer\_t method), 101  
`visit_member_operator()` (decl\_visitor\_t method), 102  
`visit_member_operator()` (linker\_t method), 147  
`visit_member_variable_type()` (linker\_t method), 147  
`visit_member_variable_type()` (type\_visitor\_t method), 134  
`visit_namespace()` (decl\_printer\_t method), 101  
`visit_namespace()` (decl\_visitor\_t method), 102  
`visit_namespace()` (linker\_t method), 147  
`visit_pointer()` (linker\_t method), 147  
`visit_pointer()` (type\_visitor\_t method), 134  
`visit_reference()` (linker\_t method), 147  
`visit_reference()` (type\_visitor\_t method), 135  
`visit_restrict()` (linker\_t method), 147  
`visit_restrict()` (type\_visitor\_t method), 135  
`visit_short_int()` (linker\_t method), 147  
`visit_short_int()` (type\_visitor\_t method), 135  
`visit_short_unsigned_int()` (linker\_t method), 147  
`visit_short_unsigned_int()` (type\_visitor\_t method), 135  
`visit_signed_char()` (linker\_t method), 147  
`visit_signed_char()` (type\_visitor\_t method), 135  
`visit_typedef()` (decl\_printer\_t method), 101  
`visit_typedef()` (decl\_visitor\_t method), 102  
`visit_typedef()` (linker\_t method), 147  
`visit_uint128()` (linker\_t method), 147  
`visit_uint128()` (type\_visitor\_t method), 135  
`visit_unsigned_char()` (linker\_t method), 147  
`visit_unsigned_char()` (type\_visitor\_t method), 135  
`visit_unsigned_int()` (linker\_t method), 147  
`visit_unsigned_int()` (type\_visitor\_t method), 135  
`visit_variable()` (decl\_printer\_t method), 101  
`visit_variable()` (decl\_visitor\_t method), 102  
`visit_variable()` (linker\_t method), 147  
`visit_void()` (linker\_t method), 147

visit\_void() (type\_visitor\_t method), [135](#)  
visit\_volatile() (linker\_t method), [147](#)  
visit\_volatile() (type\_visitor\_t method), [135](#)  
visit\_wchar() (linker\_t method), [147](#)  
visit\_wchar() (type\_visitor\_t method), [135](#)  
void\_t (class in pygccxml.declarations.cpptypes), [98](#)  
volatile\_t (class in pygccxml.declarations.cpptypes), [99](#)

## W

was\_hit (record\_t attribute), [142](#)  
wchar\_t (class in pygccxml.declarations.cpptypes), [99](#)  
we\_depend\_on\_them() (dependency\_info\_t static method), [75](#)  
working\_directory (parser\_configuration\_t attribute), [140](#)  
working\_directory (xml\_generator\_configuration\_t attribute), [140](#)  
writer (decl\_printer\_t attribute), [101](#)

## X

xml\_generator (parser\_configuration\_t attribute), [140](#)  
xml\_generator (xml\_generator\_configuration\_t attribute), [140](#)  
xml\_generator\_configuration\_t (class in pygccxml.parser.config), [140](#)  
xml\_generator\_from\_xml\_file (ietree\_scanner\_t attribute), [146](#)  
xml\_generator\_from\_xml\_file (project\_reader\_t attribute), [150](#)  
xml\_generator\_from\_xml\_file (scanner\_t attribute), [152](#)  
xml\_generator\_from\_xml\_file (source\_reader\_t attribute), [153](#)  
xml\_generator\_from\_xml\_file (xml\_generator\_configuration\_t attribute), [141](#)  
xml\_generator\_path (xml\_generator\_configuration\_t attribute), [141](#)  
xml\_generators (class in pygccxml.utils.xml\_generators), [156](#)  
xml\_output\_version (xml\_generators attribute), [157](#)