Wrapping C and C++ Libraries with CastXML

Brad King[1], Bill Hoffman[1], Matt McCormick[1], and Michka Popoff

[1] Kitware, Inc.

SciPy, 2015

Outline

1 Motivation and History
- Wrapping the Insight Toolkit (ITK)
- Generating an Abstract Syntax Tree (AST)
- GCC-XML

2 CastXML Description
- Building
- Invocation

3 SciPy Ecosystem Integration
- pygccxml

Outline

1 Motivation and History
- Wrapping the Insight Toolkit (ITK)
- Generating an Abstract Syntax Tree (AST)
- GCC-XML

2 CastXML Description
- Building
- Invocation

3 SciPy Ecosystem Integration
- pygccxml

The Insight Toolkit (ITK)

- In 1999, the US National Institute of Health's (NIH) National Library of Medicine (NLM) started a project to support the Visible Human Project.

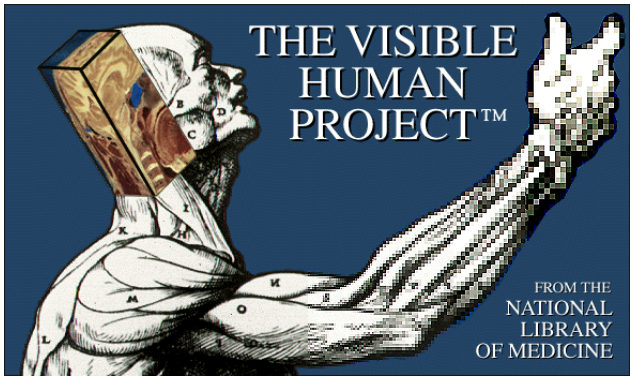
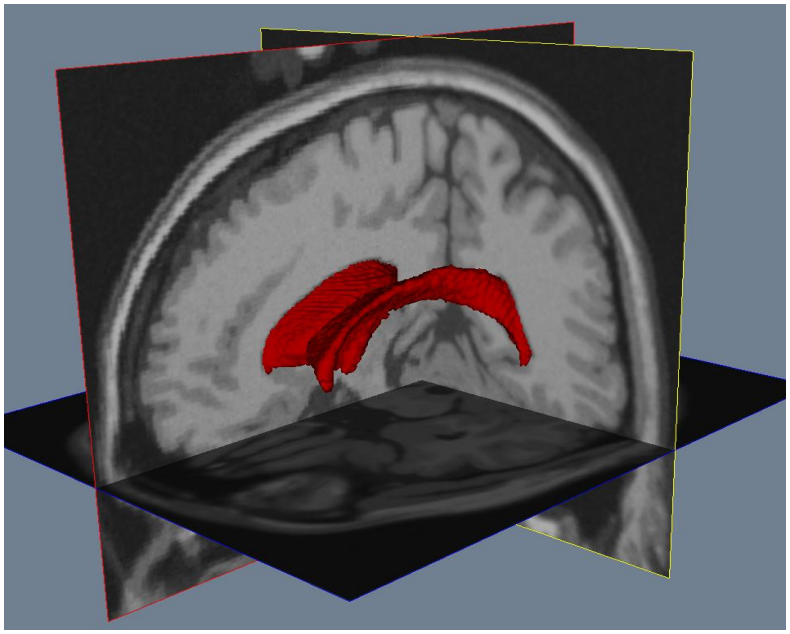


Image Segmentation



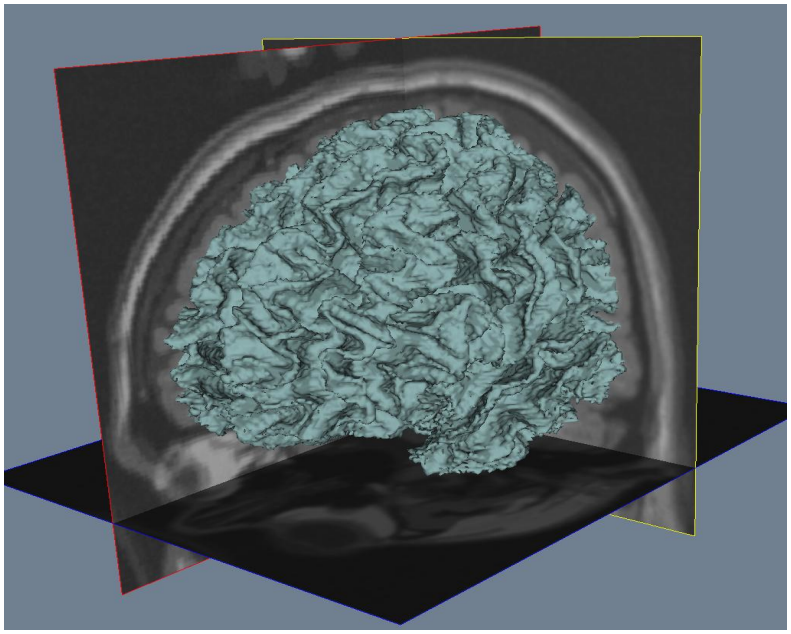
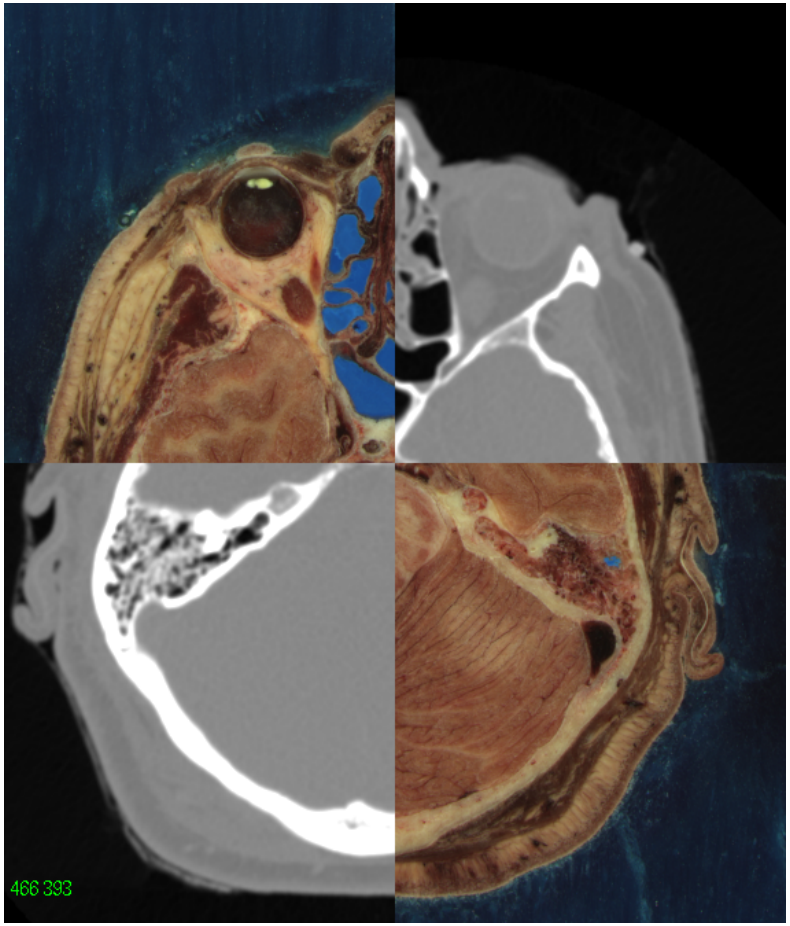
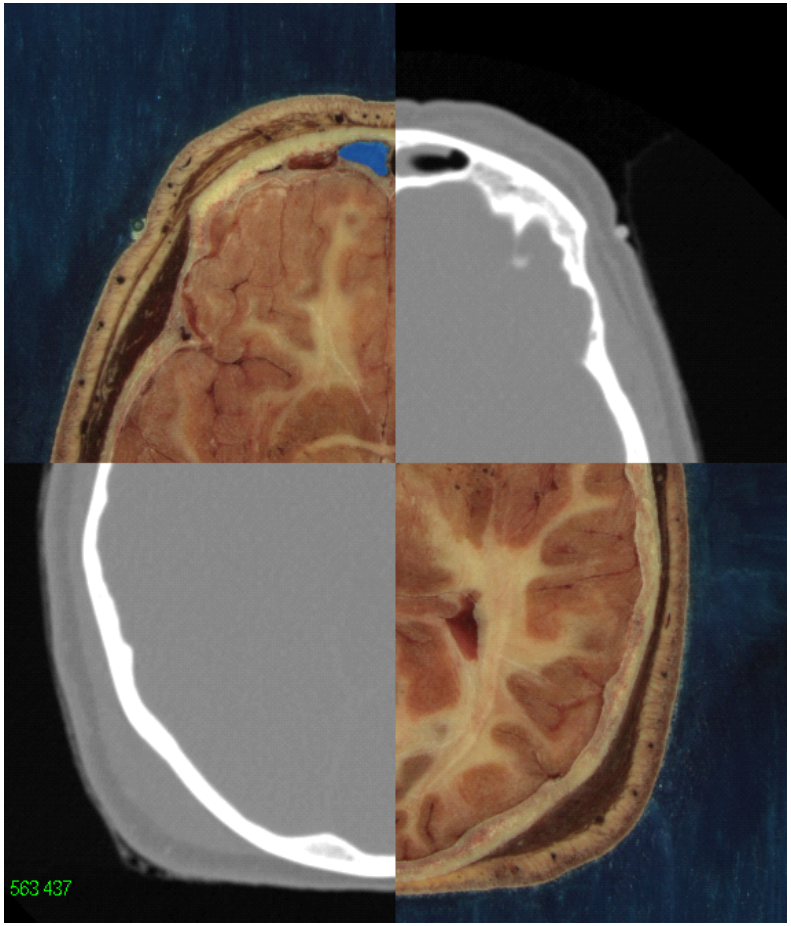


Image Registration





Large C++ Codebase

- Development has progressed since 1999
- Over 1.6 million lines of code
- Highly templated C++

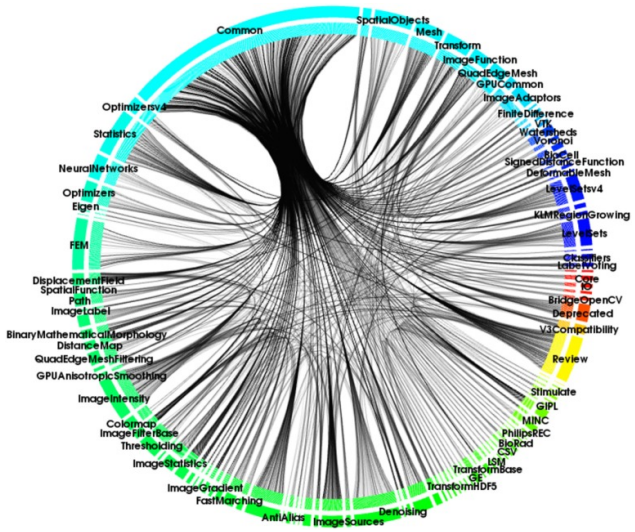


Figure: Visualization of the toolkit modules.

Large C++ Codebase

- Development has progressed since 1999
- Over 1.6 million lines of code
- Highly templated C++

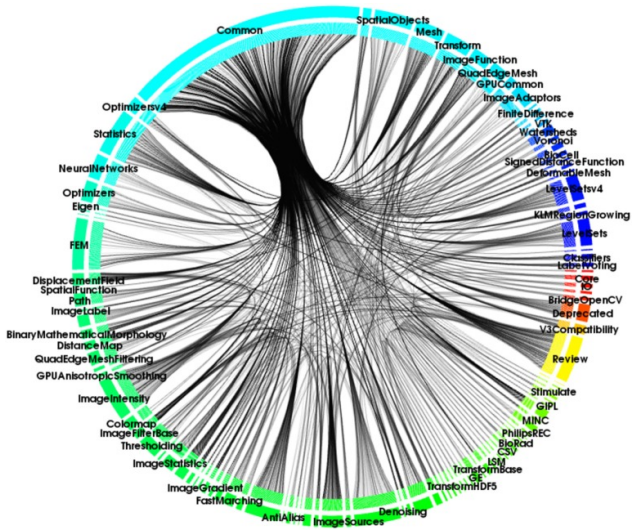


Figure: Visualization of the toolkit modules.

Large C++ Codebase

- Development has progressed since 1999
- Over 1.6 million lines of code
- Highly templated C++

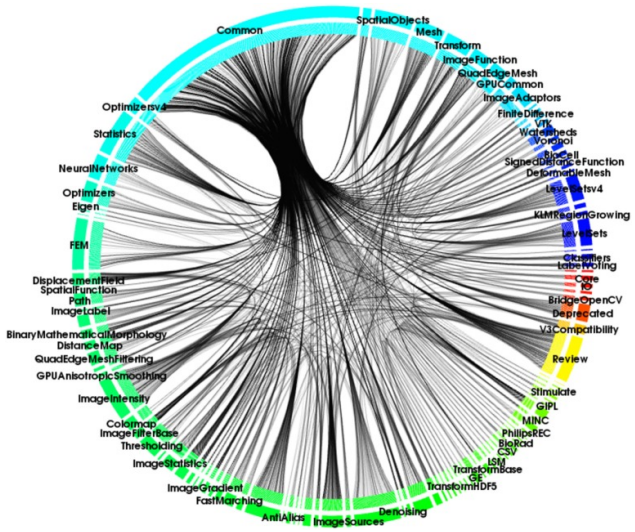


Figure: Visualization of the toolkit modules.

We Want to Use Python!

The most efficient way of 'wrapping' a big library is eating it.

ITK's Wrapping Process

1. Determine an Abstract Syntax Tree (AST) from C++ sources
2. Create Swig `.i` from the AST and desired template parameters
3. Generate CPython extension modules code with Swig
4. Build the extension modules

ITK's Wrapping Process

1. Determine an Abstract Syntax Tree (AST) from C++ sources
2. Create Swig `.i` from the AST and desired template parameters
3. Generate CPython extension modules code with Swig
4. Build the extension modules

ITK's Wrapping Process

1. Determine an Abstract Syntax Tree (AST) from C++ sources
2. Create Swig `.i` from the AST and desired template parameters
3. Generate CPython extension modules code with Swig
4. Build the extension modules

ITK's Wrapping Process

1. Determine an Abstract Syntax Tree (AST) from C++ sources
2. Create Swig `.i` from the AST and desired template parameters
3. Generate CPython extension modules code with Swig
4. Build the extension modules

ITK's Wrapping Process

1. Determine an Abstract Syntax Tree (AST) from C++ sources
2. Create Swig `.i` from the AST and desired template parameters
3. Generate CPython extension modules code with Swig
4. Build the extension modules

Outline

1 Motivation and History
- Wrapping the Insight Toolkit (ITK)
- Generating an Abstract Syntax Tree (AST)
- GCC-XML

2 CastXML Description
- Building
- Invocation

3 SciPy Ecosystem Integration
- pygccxml

C++ Source Code to Abstract Syntax Tree

```
namespace MyCoolProject {
  struct test_results {
    enum status {
      ok,
      fail,
      error
    };
    void update(
      const char * test_name,
      status result );
  };
}
```

```
<?xml version="1.0"?>
<GCC_XML version="0.9.0" cvs_revision="
  <Namespace id="_5" name="MyCoolProje
  <ArrayType id="_8" min="0" max="0" t
  <Struct id="_9" name="test_results" 
  <Typedef id="_10" name="__va_list_ta
  <Enumeration id="_11" name="status" 
    <EnumValue name="ok" init="0"/>
    <EnumValue name="fail" init="1"/>
    <EnumValue name="error" init="2"/>
  </Enumeration>
  <Method id="_12" name="update" retur
    <Argument name="test_name" type="_
    <Argument name="result" type="_11"
  </Method>
  ...
</GCC_XML>
```

C++ Source Code to Abstract Syntax Tree

```
namespace MyCoolProject {
  struct test_results {
    enum status {
      ok,
      fail,
      error
    };
    void update(
      const char * test_name,
      status result );
  };
}
```

→ Process and parse

```
<?xml version="1.0"?>
<GCC_XML version="0.9.0" cvs_revision="
  <Namespace id="_5" name="MyCoolProje
  <ArrayType id="_8" min="0" max="0" t
  <Struct id="_9" name="test_results" 
  <Typedef id="_10" name="__va_list_ta
  <Enumeration id="_11" name="status" 
    <EnumValue name="ok" init="0"/>
    <EnumValue name="fail" init="1"/>
    <EnumValue name="error" init="2"/>
  </Enumeration>
  <Method id="_12" name="update" retur
    <Argument name="test_name" type="_
    <Argument name="result" type="_11"
  </Method>
  ...
</GCC_XML>
```

C++ Source Code to Abstract Syntax Tree

```
namespace MyCoolProject {
  struct test_results {
    enum status {
      ok,
      fail,
      error
    };
    void update(
      const char * test_name,
      status result );
  };
}
```

Process and parse

```
<?xml version="1.0"?>
<GCC_XML version="0.9.0" cvs_revision="
  <Namespace id="_5" name="MyCoolProje
  <ArrayType id="_8" min="0" max="0" t
  <Struct id="_9" name="test_results" 
  <Typedef id="_10" name="__va_list_ta
  <Enumeration id="_11" name="status" 
    <EnumValue name="ok" init="0"/>
    <EnumValue name="fail" init="1"/>
    <EnumValue name="error" init="2"/>
  </Enumeration>
  <Method id="_12" name="update" retur
    <Argument name="test_name" type="_
    <Argument name="result" type="_11"
  </Method>
  ...
</GCC_XML>
```

C++ Source Code to Abstract Syntax Tree

```
namespace MyCoolProject {
  struct test_results {
    enum status {
      ok,
      fail,
      error
    };
    void update(
      const char * test_name,
      status result );
  };
}
```

Process and parse

```
<?xml version="1.0"?>
<GCC_XML version="0.9.0" cvs_revision="
  <Namespace id="_5" name="MyCoolProje
  <ArrayType id="_8" min="0" max="0" t
  <Struct id="_9" name="test_results"
  <Typedef id="_10" name="__va_list_ta
  <Enumeration id="_11" name="status"
    <EnumValue name="ok" init="0"/>
    <EnumValue name="fail" init="1"/>
    <EnumValue name="error" init="2"/>
  </Enumeration>
  <Method id="_12" name="update" retur
    <Argument name="test_name" type="_
    <Argument name="result" type="_11"
  </Method>
  ...
</GCC_XML>
```

The C++kie Model

C++ Source Code

The C++kie Model

C++ Source Code

Abstract Syntax Tree

The C++kie Model

C++ Source Code

Abstract Syntax Tree

Python Wrappers

AST Generation Complexity

C++ Source Code

- C preprocessor
- Standard library headers (compiler specific)
- Compiler flags

$\longrightarrow$

AST Generation Complexity

C++ Source Code

→

Processing

- C preprocessor
- Standard library headers (compiler specific)
- Compiler flags

→

AST Generation Complexity

C++ Source Code

$\longrightarrow$

Processing

- C preprocessor
- Standard library headers (compiler specific)
- Compiler flags

$\longrightarrow$

AST Generation Complexity

C++ Source Code

$\longrightarrow$

Processing

- C preprocessor
- Standard library headers (compiler specific)
- Compiler flags

$\longrightarrow$

AST Generation Complexity

C++ Source Code

$\longrightarrow$

Processing

- C preprocessor
- Standard library headers (compiler specific)
- Compiler flags

$\longrightarrow$

AST Generation Complexity

C++ Source Code

$\longrightarrow$

Processing

- C preprocessor
- Standard library headers (compiler specific)
- Compiler flags

$\longrightarrow$

Abstract Syntax Tree

Outline

1 Motivation and History
- Wrapping the Insight Toolkit (ITK)
- Generating an Abstract Syntax Tree (AST)
- GCC-XML

2 CastXML Description
- Building
- Invocation

3 SciPy Ecosystem Integration
- pygccxml

GCC-XML

<GCC_XML

description="XML output for GCC"/>

- Does C preprocessing
- Uses standard library headers of the simulated compiler
- Uses compiler flags for simulated compiler
- Uses GCC's C++ parsing capabilities
- Generates XML AST representation

GCC-XML

<GCC_XML

description="XML output for GCC"/>

- Does C preprocessing
- Uses standard library headers of the simulated compiler
- Uses compiler flags for simulated compiler
- Uses GCC's C++ parsing capabilities
- Generates XML AST representation

GCC-XML

<GCC_XML

description="XML output for GCC"/>

- Does C preprocessing
- Uses standard library headers of the simulated compiler
- Uses compiler flags for simulated compiler
- Uses GCC's C++ parsing capabilities
- Generates XML AST representation

GCC-XML

<GCC_XML

description="XML output for GCC"/>

- Does C preprocessing
- Uses standard library headers of the simulated compiler
- Uses compiler flags for simulated compiler
- Uses GCC's C++ parsing capabilities
- Generates XML AST representation

GCC-XML

<GCC_XML

description="XML output for GCC"/>

- Does C preprocessing
- Uses standard library headers of the simulated compiler
- Uses compiler flags for simulated compiler
- Uses GCC's C++ parsing capabilities
- Generates XML AST representation

GCC-XML

<GCC_XML

description="XML output for GCC"/>

- Does C preprocessing
- Uses standard library headers of the simulated compiler
- Uses compiler flags for simulated compiler
- Uses GCC's C++ parsing capabilities
- Generates XML AST representation

General Projects That Also Used GCC-XML

- Pyste (Boost.Python)
- Py++ (Boost.Python)
- Python ctypeslib
- xdress
- Others...

General Projects That Also Used GCC-XML

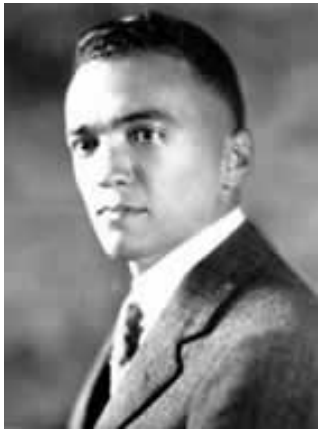
- Pyste (Boost.Python)
- Py++ (Boost.Python)
- Python ctypeslib
- xdress
- Others...

General Projects That Also Used GCC-XML

- Pyste (Boost.Python)
- Py++ (Boost.Python)
- Python ctypeslib
- xdress
- Others...

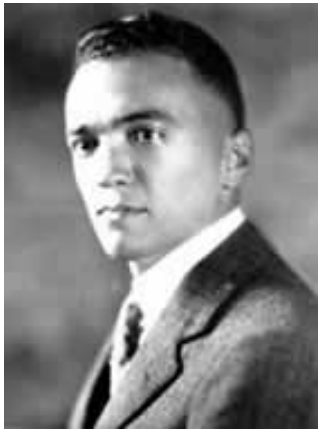
General Projects That Also Used GCC-XML

- Pyste (Boost.Python)
- Py++ (Boost.Python)
- Python ctypeslib
- xdress
- Others...



General Projects That Also Used GCC-XML

- Pyste (Boost.Python)
- Py++ (Boost.Python)
- Python ctypeslib
- xdress
- Others...



GCC-XML's Weakness

GCC-XML's Weakness

Maintainability.

CastXML

CastXML is a GCC-XML replacement that uses LLVM/Clang.

CastXML

A **C**-family **a**bstract **s**yntax **t**ree **XML** output tool.

Outline

1 Motivation and History
- Wrapping the Insight Toolkit (ITK)
- Generating an Abstract Syntax Tree (AST)
- GCC-XML

2 CastXML Description
- Building
- Invocation

3 SciPy Ecosystem Integration
- pygccxml

CastXML

Building requires

- A C++ compiler with C++11 support
- CMake
- LLVM/Clang 3.6.0 or later. Currently must be
  - Built from source
  - Installed into an install prefix

CastXML

Building requires

- A C++ compiler with C++11 support
- CMake
- LLVM/Clang 3.6.0 or later. Currently must be
  - Built from source
  - Installed into an install prefix

CastXML

Building requires

- A C++ compiler with C++11 support
- CMake
- LLVM/Clang 3.6.0 or later. Currently must be
  - Built from source
  - Installed into an install prefix

Invocation

Outline

1 Motivation and History
- Wrapping the Insight Toolkit (ITK)
- Generating an Abstract Syntax Tree (AST)
- GCC-XML

2 CastXML Description
- Building
- Invocation

3 SciPy Ecosystem Integration
- pygccxml

Invocation

```
castxml \
  --castxml-gccxml \
  --castxml-cc-msvc cl \
  -o ouput.xml
  input.cxx
```

Outline

1 Motivation and History
- Wrapping the Insight Toolkit (ITK)
- Generating an Abstract Syntax Tree (AST)
- GCC-XML

2 CastXML Description
- Building
- Invocation

3 SciPy Ecosystem Integration
- pygccxml

pygccxml

pygccxml

A specialized XML reader that reads the output from CastXML and GCC-XML. It provides a simple framework to navigate C++ declarations using Python classes.

Repository: `https://github.com/gccxml/pygccxml`

pygccxml History

- The original author is Roman Yakovenko (2004-2011).
- Forked multiple times by different authors to add Python 3 support.
- In May 2014, Michka Popoff and the Insight Software Consortium revived pygccxml.

pygccxml History

- The original author is Roman Yakovenko (2004-2011).
- Forked multiple times by different authors to add Python 3 support.
- In May 2014, Michka Popoff and the Insight Software Consortium revived pygccxml.

pygccxml History

- The original author is Roman Yakovenko (2004-2011).
- Forked multiple times by different authors to add Python 3 support.
- In May 2014, Michka Popoff and the Insight Software Consortium revived pygccxml.

New since 2014

- Mercurial to Git
- Travis-ci.org hook for Python 2.6, 2.7, 3.2, 3.3, 3.4 support.
- coverall.io hook for code coverage
- Docs on readthedocs.org
- quantifiedcode.com hook for code structure
- PEP8 compatible

pygccxml: API example

Input

```
namespace MyCoolProject {
  struct test_results {
    enum status {
      ok,
      fail,
      error
    };
    void update(
      const char * test_name,
      status result );
  };
}
```

Output

```
Name: update
Return type: void
Argument 0: char const * test_name
Argument 1: MyCoolProject::test_results::status result
```

Python

```
from pygccxml import parser
from pygccxml import declarations
# Snip --//-- Setup config object for castxml or gccxml
decls = parser.parse(["test.cpp"], config)
global_ns = declarations.get_global_namespace(decls)
project_ns = global_ns.namespace("MyCoolProject")
update_functions = project_ns.member_functions("update")
# There could be multiple functions with the same name (in c++)
for function in update_functions:
    print("Name: " + function.name)
    print("Return type: " + str(function.return_type))
    for i, arg in enumerate(function.arguments):
        print("Argument " + str(i) + ": " + str(arg))
```

Summary

- Generating an Abstract Syntax Tree is an important part of the wrapping process.
- The Abstract Syntax Tree is compiler-dependent because of the C preprocessor, standard library headers, and lines.
- CastXML captures the compiler-dependence and does complex parsing of C++.
- pygccxml is a Python interface to CastXML output

Resources I

Mailing list

http://public.kitware.com/mailman/listinfo/castxml

Repository

https://github.com/CastXML/CastXML