Parsing Documentation Comments in Clang

Dmitri Gribenko ⟨gribozavr@gmail.com⟩
High Performance Computing Center at NTUU «KPI»

LLVM Developers’ Meeting
November 8, 2012
Outline

- Motivation
- Implementation details
- User-visible features in Clang and libclang
Clang is (was) ignoring documentation comments

- Comments tend to bitrot.
- Syntax errors in markup are not caught.
- Documentation extraction tools don’t really understand C++.
- Tools based on Clang libraries will also ignore comments:
  - IDE source editor tools;
  - refactoring tools.
Documentation support: goals

- Help to keep comments up to date and syntactically correct.
- Enhance Clang-based tools with comments support:
  - IDE source editor tools;
  - refactoring tools.
- (Maybe) Build a better documentation extraction tool.
Documentation comments
≈
Doxygen
Implementation

- Finding comments.
- Attaching comments to declarations.
- Comment parsing.
- Comment AST.
Attaching comments to AST nodes

• Teach Clang parser about `tok::comment`:
  • would complicate the parser;
  • would add lots of checks for `tok::comment`;
  • no easy way to turn off.

• Attach comments after the AST is built.
Finding raw comments

Lexer → Parser → Sema::ActOn...

Comment Handler → Sema::ActOnComment → ASTContext::Comments

tok::comment → StringRef Source Location → RawComment
Documentation comments

/** comment */
void g();
int b; /** comment */

/*! comment */
void h();
int c; /*! comment */

/// comment
void f();
int a; /// comment
"Almost documentation" comments

```c
struct a {
    int x; //< comment
    int y; /*< comment */
};
```
"Almost documentation" comments

```c
struct a {
    int x; // /* comment */
    int y; /* comment */
};
```

tagged with the warning:

```c
example.c:8:10: warning: not a Doxygen trailing comment
    int x; // comment
    ^~~
    ///<
```
Attaching comments to AST nodes

```c
/**
 * ...
 */

void doSomething();

///
///...
///

void doSomethingElse();

void doSomethingInteresting();
```
Attaching comments to AST nodes

```java
/**
 * ...
 */

void doSomething();

///
///...
///

void doSomethingElse();

void doSomethingInteresting();
```
Attaching comments to AST nodes

```java
/**
 * ...
 */

void doSomething();

///
///...
///

void doSomethingElse();

void doSomethingInteresting();
```
Attaching comments, eager mode

Sema::ActOn...

Sema::ActOnDocumentableDecl

FullComment

ASTContext::Comments

RawComment

ASTContext::getCommentForDecl

FullComment

Comment lexer, parser, sema

ASTContext::ParsedComments
Comments from "related" declarations

/// \brief Does foo.
/// \param a blah.
void f(int a);

void f(int b) { // picks up the comment above.
}
struct A {
    /// \brief Does foo.
    virtual void f();
};

struct B : public A {
    virtual void f(); // picks up A::f()'s comment.
};
Comment parsing

Comment Lexer

Full Parser

\brief Parser

Comment Sema

Brief description

AST
Comment AST

- Root `FullComment` node has children which are block content.
- `ParagraphComment` nodes have children which are inline content.
Comment AST

- Root `FullComment` node has children which are block content.
- `ParagraphComment` nodes have children which are inline content.

- Block content:
  - paragraph;
  - block command (e.g., `\brief` or `\verbatim`).

- Inline content:
  - text;
  - inline command (e.g., `\c`);
  - HTML tag.
/// \brief Does something with \p str.
/// \param [in] Str the string.
/// \returns a modified string.
void do_something(const std::string &str);
User-visible features

- Diagnostics for comments.
- Enhanced code completion APIs (libclang).
- Export comments as XML (libclang).
- Performance.
void do_something(const std::string &str);
Diagnostics

/// \brief Does something with \p str.
/// \param [in] Str the string.
/// \returns a modified string.
void do_something(const std::string &str);

example.cc:4:17: warning: parameter 'Str' not found
     in the function declaration [-Wdocumentation]
/// \param [in] Str the string.
    ^~~
example.cc:4:17: note: did you mean 'str'?
/// \param [in] Str the string.
    ^~~
    str
Diagnostics

/// brief Does something with \p str.
/// param [in] Str the string.
/// returns a modified string.
void do_something(const std::string &str);

example.cc:5:6: warning: 'eturns' command used in a comment that is attached to a function returning void [-Wdocumentation]
/// \returns a modified string.
~\~~~~~~~~~~~~~~~~~~~~~~~~~
/// \param x value of X coordinate.
/// \param x value of Y coordinate.

void do_something(int x, int y);

element.cc:2:12: warning: parameter 'x' is already documented [-Wdocumentation]
/// \param x value of Y coordinate.
^
/// \brief Does something.
/// \deprecated
void do_something();

example.cc:4:6: warning: declaration is marked with `\deprecated` command but does not have a deprecation attribute
example.cc:5:19: note: add a deprecation attribute to the declaration to silence this warning
void do_something();
__attribute__((deprecated))
Diagnostics

/// \brief Does something.
/// \deprecated
void do_something();

e#example.cc:4:6: warning: declaration is marked with
// \deprecated
\^~~~~~~~~~~~~~~~~
example.cc:5:19: note: add a deprecation attribute
to the declaration to silence this warning

void do_something();
\^__attribute__((deprecated))
#define MY_DEPRECATED __attribute__((deprecated))

/// \brief Does something.
/// \deprecated
void do_something();

example.cc:4:6: warning: declaration is marked with \deprecated command but does not have a deprecation attribute

/// \deprecated
~\~~~~~~~~~

example.cc:5:19: note: add a deprecation attribute to the declaration to silence this warning
void do_something();

^   

MY_DEPRECATED
LLVM and Clang are 99% -Wdocumentation clean.

Thanks to James Dennett!
LLVM and Clang are 99% -Wdocumentation clean.

Thanks to James Dennett!

Please configure LLVM and Clang with
"--with-extra-options=-Wdocumentation
-Wno-documentation-deprecated-sync"
and help to clean up the last bits.
Enhanced code completion

Code completion provides:

- text or patterns to insert;
- priority;
- availability information;
Enhanced code completion

Code completion provides:

- text or patterns to insert;
- priority;
- availability information;
- brief documentation.

```c
/// \param x ...
/// \param y ...
/// \brief Does foo.
/// \returns bar.
int foo(int x, int y);

/// Does foo.
/// Sometimes does bar.
int foo(int x, int y);
```
XML export

- A comment can be converted to XML.
- XML schema: \texttt{bindings/xml/comment-xml-schema.rng}.
- XML document includes:
  - parsed comment;
  - information about the declaration.
template<typename T> void foo(int x, T t);

<?xml version="1.0"?>
<Function templateKind="template" file="example.cc"
    line="5" column="6">
    <Name>foo</Name>
    <USR>c:@FT@&gt;1#Tfoo#I#t0.0#</USR>
    <Declaration>template &lt;typename T&gt;
        void foo(int x, T t)</Declaration>
    <Abstract><Para> Does quux. </Para></Abstract>
    <TemplateParameters> ... 
    <Parameters> ...
Performance

- Parse comments only if `-Wdocumentation` is passed.
- If you have lots (> 10000) of documentation comments, you might see a < 5% regression in `-fsyntax-only` time.
  - Parsing time is only a small fraction of compile time.
- Comments from system headers are skipped during normal compilation.
  - There is no sense in parsing them for diagnostics.
  - Pass `-fretain-comments-from-system-headers` if you want them back.
Demo

Vim with clang_complete
clang_complete with brief documentation support

```cpp
square(int x)
Returns the number multiplied by itself.

/// Returns the number multiplied by itself.
/// \param x input number.
int square(int x);

int square(int x) { return x * x; }
```

https://github.com/gribozavr/clang_complete
Future work

- Resolve parameter names in \p.
- Resolve declaration references.
- Attach comments to macros.
- AST matchers?
- Integrate this feature with other IDEs (maybe after clangd?)
- Write a better documentation extraction tool.
  - Fix the declaration pretty-printer.
Summary

Now Clang can:

- parse documentation comments;
- find semantic errors in comments (try `-Wdocumentation`);
- export comments as XML.

This work enables more awesome features in future!
Q&A