Documenting C++ Using the Right Tools

Barbara Geller & Ansel Sermersheim

CppNow - May 2016

Introduction

- What is the value of documentation
- Overview of Doxygen
- Why we developed DoxyPress
- Parsing C++
  - libClang
  - libTooling
- Migrating code from C++98 to C++11
- Future plans for DoxyPress

What is the Value of Documentation

- Who needs documentation
  - developers of your application
  - users of your library or application
  - your future self

- What should be documented
  - class and method documentation
  - how to set up your environment
  - process for building your application
  - overall system design
  - timeline or change log, error conditions
  - samples code

Why Documentation is Important

- When to create documentation
  - day one of your project

- Maintaining documentation
  - refer to your documentation to ensure it is accurate
  - use your own build documentation
  - the more out of sync your documentation is, the less likely you will be to update it

Overview of Doxygen

- Development started around 1995
- Open Source / GPL 2

- Uses obsolete/unmaintained Qt 1.9 classes
- Non standard language translation functionality
- Project config file is raw text, parsed with lex
- Excessive use of ternary ? : operator
- Parameters which shadow member variables

Why we Developed DoxyPress

- Unable to document CopperSpice C++ libraries
- Initial direction was to help improve Doxygen which turned out not to be feasible
- Code was simply unmaintainable

- DoxyPress is derived from Doxygen
- DoxyPress and DoxyPressApp link with the CopperSpice libraries

Problematic C++ Code

- Macros used to simulate variadic templates
- Raw pointers used exclusively
- No smart pointers

- Code extremely difficult to read
  - limited line breaks
  - prolific use of variable names like: bcli, bii, cli, cei, cni, di, dcli, ei, eli, evi, i, ii, iii, l, li, lii, lli, mli, mnii, mri, pli, sl, sli, slii

Problematic C++ Code

- Container classes are all pointer based
- Autodelete memory management

- `std:set<T>` simulated by using the equivalent of `std::map<std::string, void *>`

```
accessors->insert(s, (void *)666);
```

- Many of the internal classes inherit from containers

Problematic C++ Code

```
// QDict<T> is like std::map<std::string, T*>
class FileNameDict : public QDict<FileName>

class FileName : public FileList  {
  // contains 3 methods, 2 data members
}

// QList<T> is like std::list<T*>
class FileList : public QList<FileDef> {
  // contains 2 methods, 1 data member
  // one of the methods compares FileDef entries
}

class FileDef : public Definition
class Definition : public DefinitionIntf
class DefinitionIntf
```

Doxygen - Example 1

- Custom string class
  - result returns ‘\0’ if an invalid index is accessed
  - access off the end of a string is acceptable code

```
if (result.at(0) == ':' && result.at(1) == ':')  {
    . . .
}
```

DoxyPress - Example 1

- CopperSpice QString, similar to std::string
  - accessing an invalid index is an error

```
// A, initial fix
int len = result.size();
if (len >= 2 && result.at(0) == ':' && result.at(1) == ':') {
    . . .
}

// B, optimized
if (result.startsWith("::"))  {
    . . .
}
```

Doxygen - Example 2

- For “3” parameters there were 9 different forms

```
FORALL3(bool a1, Item a2, const char *a3, a1, a2, a3)
FORALL3(bool a1, bool a2, bool a3, a1, a2, a3)
FORALL3(const ClassDiagram &a1,const char *a2, const char *a3, a1, a2, a3)
FORALL3(const char *a1, const char *a2, const char *a3, a1, a2, a3)
FORALL3(const char *a1, const char *a2, bool a3, a1, a2, a3)
FORALL3(const char *a1, int a2,const char *a3, a1, a2, a3)
FORALL3(const char *a1, const char *a2, SectionType a3, a1 ,a2, a3)
FORALL3(uchar a1, uchar a2, uchar a3, a1, a2, a3)
FORALL3(Definition *a1, const char *a2, bool a3, a1, a2, a3)
```

Doxygen - Example 2

- Code also existed for passing X number of parameter with various data type combinations
  - FORALL6 (2 forms)
  - FORALL5 (2 forms)
  - FORALL4 (4 forms)
  - FORALL2 (9 forms)
  - FORALL1 (12 forms)
- Over 200+ lines of code

Doxygen - Example 2

- FORALL3() is a macro used to forward 3 parameters to a method

```
#define FORALL3(a1,a2,a3,p1,p2,p3) \
void OutputList::forall(void (OutputGenerator::*func)(a1,a2,a3), \
     a1,a2,a3) \
{ \
    QListIterator<OutputGenerator> it(m_outputs); \
    OutputGenerator *og; \
    for (it.toFirst();og=it.current();++it) \
    { \
        if (og->isEnabled()) (og->*func)(p1,p2,p3); \
    } \
}
```

DoxyPress - Example 2

- The entire FORALL macros were replaced with the following 9 lines of code

```
template<class BaseClass, class... Args, class... Ts>
void forall(void (BaseClass::*func)(Args...), Ts &&... Vs)
{
    for (auto item : m_outputs) {
        if (item->isEnabled()) {
            (item->*func)(Vs...);
        }
    }
}
```

Parsing C++

Parsing

- Parsing is normally done in multiple phases (1)
  - Lexical Analysis -- Lex
    - groups the input stream into a set of tokens
      - identifiers, keywords, literals, punctuation, etc
    - tokenizing produces a stream of tokens
    - regular expressions are used to define the lexical patterns
  - Lex is a tool for generating a scanner which can recognizes lexical patterns in a text stream and produce a stream of tokens

Parsing

- Parsing is normally done in multiple phases (2)
  - Semantic Parsing -- Bison
    - tokens are parsed to discover the structure of the source code
    - during the parsing process an Abstract Syntax Tree (AST) is created
    - an AST reflects the syntactical structure of your readable code into a tree structure
  - Bison is a tool to generate a parser, a program which recognizes the grammatical structure of your source code

Parsing

- C++ parsing in Doxygen
  - entirely implemented using Lex
  - Lex is used for both the lexical phase and the semantic analysis phase
  - a single Lex parser is used for these languages:
    - C, C++, C#, Objective-C, D, IDL, Java, JS, and PHP
  - approximately 800 different rules
  - many of the rules resolve different languages
  - not always clear which rules are for which languages

Parsing Multiple Programming Languages

```
<ClassVar>{ID}    {
  QString text = QString::fromUtf8(yytext);

  if (insideIDL && text == "switch") {

  } else if ((insideJava || insidePHP || insideJS) &&
             (text == "implements" || text =="extends") ) {

  } else if (insideCSharp && text == "where") {

  } else if (insideCli && text == "abstract") {

  } else if (insideCli && text == "sealed") {

  } else if (text == "final") {

  } else {

  // ...
```

Parsing Rules

```
BN              [ \t\n\r]
ID              "$"?[a-z_A-Z\x80-\xFF][a-z_A-Z0-9\x80-\xFF]*
TYPEDEFPREFIX   (("typedef"{BN}+)?)((("volatile"|"const"){BN}+)?)


<SkipCurly>"}"/{BN}*("/*!"|"/**"|"//!"|"///")"<!--" |
<SkipCurly>"}"
    // parsing comments in source

<FindMembers>{B}*{TYPEDEFPREFIX}{IDLATTR}?"enum"({BN}+("class"|"struct"))?"{" |
<FindMembers>{B}*{TYPEDEFPREFIX}{IDLATTR}?"enum"({BN}+("class"|"struct"))?{BN}+
    //

<FindMembers>{BN}*(("disp")?"interface")|"valuetype"){BN}+
    // M$/Corba/UNO IDL/Java interface

<EndTemplate>">"{BN}*/"("({BN}*{ID}{BN}*"::")*({BN}*"*"{BN}*)+
    // function pointer returning a template instance
```

Parsing Problems

- Since multiple languages are parsed from this one Lex file, any changes can introduce multiple bugs
- New rules must be added to this parser each time any language is enhanced
- Lex does not handle look ahead expressions well
- For CopperSpice we had to add about 50 new rules
- How do you stay current with C++17 and C++20?

DoxyPress C++ Parsing

- Our approach was to use libClang
  - libClang is a C Interface to Clang
  - provides a relatively small API
  - exposes functionality for parsing source code into an abstract syntax tree (AST)

  - used by XCode
  - syntax highlighting
  - code completion

DoxyPress C++ Parsing

- libClang
  - parse a file to generate the “cursors”
  - traverse the AST
  - associate locations in source with elements in the AST

  - libClang was not designed to provide all of the information in Clang's C++ AST
  - the intent of libClang is to maintain an API that is relatively stable from one release to the next and provide only the basic functionality needed to support development tools

Parsing C++ / libClang

- Parse C++ (A)
  - initial setup and configuration
  - obtain the translation unit (TU) for the given source file
  - check for syntax errors
  - walk the AST and visit all cursors, recursively
    - match on a type of cursor
      - declaration, enum, class, method, members, etc
  - save the cursor attributes

- Locate the comments (B)
  - generate the tokens for a TU
  - generate the cursors for each token
  - walk the tokens looking for comments

Parsing C++ / libClang

- Cursor
  - represents a location within the AST
  - libClang has methods to map between cursors and the physical locations where the entities occur in the source
- Token
  - smallest element of a program which is meaningful to the compiler
  - identifiers, keywords, literals, operators, separators

Parsing C++ / libClang

- Match on cursor kinds (A)

```
friend int kayakCapacity ( int len , int width ) ;
```

  - in libClang this has a CXCursorKind of CXCursor_CXXMethod
  - we save the appropriate data in class Entry in DoxyPress to simulate what was saved in the original Lex parser

- Comments (B)
  - locate a comment by testing all the tokens
  - add the comment to an existing Entry object

Parsing C++ / libClang

- The next seven slides contain source code from “parser_clang.cpp” in DoxyPress

- The code in these slides has been condensed for readability and to show the most meaningful lines

Parsing C++ / libClang

```
// obtain the Translation Unit

class ClangParser::Private {
   CXIndex index;
   CXTranslationUnit tu;
   CXCursor *cursors;
   CXUnsavedFile *ufs;
}

uint numUnsavedFiles;


CXErrorCode errorCode = clang_parseTranslationUnit2(p->index, 0,
      argv, argc, p->ufs, numUnsavedFiles,
      CXTranslationUnit_DetailedPreprocessingRecord, &(p->tu) );
```

Parsing C++ / libClang

```
// walk the AST and visit all cursors, recursively


// top of the cpp
static QSharedPointer<Entry> s_current_root;


// obtain tu
s_current_root = root;

CXCursor rootCursor = clang_getTranslationUnitCursor(p->tu);
clang_visitChildren(rootCursor, visitor, nullptr);
```

Parsing C++ / libClang

```
// call back, called for each cursor node

static CXChildVisitResult visitor(CXCursor cursor,
      CXCursor parentCursor, CXClientData clientData)
{
  . . .

  CXCursorKind kind = clang_getCursorKind(cursor);
  QSharedPointer<Entry> parentEntry;

  switch (kind) {
    // multiple cases
  }

  return CXChildVisit_Recurse;
}
```

Parsing C++ / libClang

```
case CXCursor_FunctionDecl:

  QString signature = getCursorDisplayName(cursor);
  QString name = getCursorSpelling(cursor);
  QString args = signature.mid(name.length());

  QSharedPointer<Entry> current = QMakeShared<Entry>();

  current->section = Entry::FUNCTION_SEC;
  current->name = name;
  current->type = getCursorResultType(cursor);
  current->args = args;

  QString key = getCursorUSR(cursor);
  s_entryMap.insert(key, current);

  break;
```

Parsing C++ / libClang

```
case CXCursor_CXXBaseSpecifier:
  QString name = getCursorSpelling(cursor);

  if (s_lastClassEntry != nullptr && ! name.isEmpty()) {
    Protection protection = getAccessSpecifier(cursor);

    Specifier virtualType = Specifier::Normal;
    if (clang_isVirtualBase(cursor)) {
      virtualType = Specifier::Virtual;
    }

    if (name.startsWith("class ") ) {
       name = name.mid(6);
    } else if (name.startsWith("struct ") ) {
       name = name.mid(7);
    }
    // inheritance, save class name & virtualType to the parent Entry
  }
```

Parsing C++ / libClang

```
case CXCursor_CXXMethod:
case CXCursor_FunctionTemplate:

  QSharedPointer<Entry> current = QMakeShared<Entry>();

  if (clang_CXXMethod_isPureVirtual(cursor)) {
    current->type.prepend(" virtual ");
    current->virt = Specifier::Pure;
    tmpArgs += " = 0";
    tmpList.pureSpecifier = true;

  } else if (clang_CXXMethod_isVirtual(cursor)) {
    current->type.prepend(" virtual ");
    current->virt = Specifier::Virtual;

  }

  // . . .
```

Parsing C++ / libClang

```
CXToken *tokens;
uint numTokens;

clang_tokenize(p->tu, range, &tokens, &numTokens);

for (int j = 0; j < numTokens - 1; j++) {
   QString text = getTokenSpelling(p->tu, tokens[j]);

  if (text == "(") {
    break;

  } else if (text == "constexpr") {
    current->type.prepend("constexpr ");

  } else if (text == "inline") {
    current->m_traits.setTrait(Entry::Virtue::Inline);
}
```

Parsing C++ / libClang

- libClang wrappers are missing or do not work correctly when parsing a method
  - default values
  - constexpr, explicit, inline
  - delete, default, final, noexcept, volatile
- Friend declarations do not work at all
  - walking the tokens for this cursor kind and parsing the declaration works, except for the argument list

Parsing C++

- When the documentation said libClang was missing a “few” parts of the AST, they really meant...
  - libClang is maintained by a few users
  - it is a C interface and not intended for C++ parsing
  - used for XCode, almost no one else is using it
  - use Clang if you need full parsing

- What we gained
  - how to traverse and understand the AST
  - how to store the parsed information in an Entry to generate documentation

Parsing C++ / LibTooling

- Create a few classes which inherit from
  - clang::RecursiveASTVisitor
  - clang::ASTConsumer
  - clang::ASTFrontendAction

```
class DoxyVisitor : public RecursiveASTVisitor<DoxyVisitor> {
  // . . .
  bool VisitCXXRecordDecl(CXXRecordDecl *node) override { . . . }
  bool VisitFunctionDecl(FunctionDecl *node) override   { . . . }
  // . . .
}
```

Parsing C++ / LibTooling

- Usually a libTooling project is located in the llvm source tree
- Deciphering include files
  - resolved by trial and error
- Deciphering lib files
  - complicated

Migrating to modern C++

Migrating from C++98 to C++11

- Ensure copy constructor is a deep copy

- Raw pointers ➡ shared pointers
  - with raw pointers it is unclear who is responsible for object destruction
  - too easy to accidentally use a raw pointer after the object has been deleted
  - use QMakeShared in CopperSpice or std::make_shared instead of calling new

  - this type of pointer conversion can not be done gradually

Migrating from C++98 to C++11

- for loop
  - C++11 range based syntax
  - use auto for declaring iterators

- Container misuse
  - `QHash<QString, void *> files;`
  - `files.insert("myFile", (void *)0x08);`
  - a large amount of code used raw pointers

- Override
  - ensure methods which override a base class method are marked with “override”

Migrating from C++98 to C++11

- Character set encoding
  - use UTF-8 internally
  - program as if your application will be used internationally
- Strings
  - avoid using const char * (memory management issues)
  - use std::string class, or
  - use QString class in CopperSpice
- Use nullptr instead of 0
  - improves readability
  - zero can mean nullptr or an empty string

Future Plans

Where DoxyPress is At

- Removed all Qt 1.9 classes and containers
- Code reformatted
- Enhanced source to use C++11
- Using shared pointers instead of raw pointers
- Variadic templates instead of macro abuse

- Project file changed from raw text to JSON format
- DoxyPressApp converts a Doxygen project file to a DoxyPress project file

Future Plans for DoxyPress

- Complete integration with clang for parsing C++
- Redesign internal containers
- Update memory model
- Support for other languages like D
- User requests & developer contributions

Libraries & Applications

- CopperSpice
  - libraries for developing GUI applications
- PepperMill
  - converts Qt headers to CS standard C++ header files
- CsSignal Library
  - standalone thread aware signal / slot library
- LibGuarded
  - standalone multithreading library for shared data

Libraries & Applications

- KitchenSink
  - one program which contains 30 demos
  - links with almost every CopperSpice library

- Diamond
  - programmers editor which uses the CS libraries

- DoxyPress & DoxyPressApp
  - an application for generating documentation

Where to find our libraries

- www.copperspice.com
- download.copperspice.com
- forum.copperspice.com

- ansel@copperspice.com
- barbara@copperspice.com

- Questions? Comments?