



GSoC Report: Clang Integration in KDevelop

Kevin Funk (kfunk@kde.org)

September 6, 2014 | Brno | Academy 2014



History of KDevelop

- Issues with current C++ Support
 - **Custom parser** living inside KDevelop code base
 - Over **50 000 LOC**
 - **Hard to maintain**, even harder to extend
 - Hint: C++11, C++14, ...
 - Lots of issues with non-trivial C/C++ code
 - (designated initializers, ...)
 - Not possible to disambiguate between C vs. C++, or separate C++ standard versions
 - ...



Clang to the rescue!

- C/C++/ObjC language frontend for LLVM
- Features
 - Expressive diagnostics
 - Allows tight integration with IDEs
 - (fuzzy parsing, incremental compilation)
 - BSD-licensed
 - Highly active community
 - Stable API via **libclang**





Clang Integration in KDevelop

- Our hopes
 - No need to maintain a custom C++ parser
 - Let Clang handle parsing + semantic analysis
 - Get refactoring tools for free
 - Get ObjectiveC support for free
- Our intend
 - Enables us to focus on **improving the IDE**, not **the parser!**



GSoC Work: Outline

Screenshots!



GSoC Work: Assistants

- Providing diagnostics/fixits from Clang

A screenshot of the KDevelop IDE showing a C++ code editor window titled 'main.cpp'. The code is as follows:

```
1 int main()
2 {
3   int foo
4 }
5
```

The line containing 'int foo' is highlighted in red. At the bottom of the editor, a 'Fix-it Hints' panel is visible, showing a single hint: '1 Insert ";" at line: 2, column: 11'. To the right of the hint is a '0 Hide' button. The status bar at the top right of the editor indicates 'Line: 3 Col: 12'.



GSoC Work: Code completion

- Virtual override completion [ported]

```
1 struct Base {  
2     virtual void foo() {}  
3 };  
4  
5 struct Derived : Base  
6 {  
7     ...
```

Virtual Override
Override void foo()
Global
Base
Derived
Builtin
auto
bool
char
char16_t
char32_t



GSoC Work: Code completion

- Implement Function Helper [ported]

The screenshot shows the KDevelop IDE with a file named 'main.cpp' open. The code in the editor is:

```
1 struct Foo
2 {
3     void foo();
4 };
5
6
```

A context menu is open over the code, showing the following options:

- Implement Function
- Implement void ← Foo::foo() (highlighted)
- Global
- Builtin
 - asm(string-literal)
 - auto



GSoC Work: Code completion

- Show viable expressions for current context

The screenshot shows a code editor window titled 'main.cpp' with the following code:

```
1 #include <cstring>
2
3 int main()
4 {
5     int i = 1; const char* str = "c";
6     strlen();
```

The cursor is at the end of the `strlen()` call on line 6, column 12. A dropdown menu is open, showing the following suggestions:

- Best matches
 - const char * str
 - const char * basename(const char * __filename)
 - const char * index(const char * __s, int __c)
 - const char * rindex(const char * __s, int __c)
 - const char * strcasestr(const char * __haystack, const char * __needle)
 - const char * strchr(const char * __s, int __c)
- Local
 - int i
 - const char * str
- Global
 - char * basename(char * __filename)



GSoC Work: Code completion

- Special completion: Enum-case labels

```
1 enum SomeEnum { aaa, bbb };
2
3 enum AnotherEnum { ccc, ddd };
4
5 int foo, bar;
6
7 int main()
8 {
9     SomeEnum e;
10    switch (e) {
11    case |
```

The completion popup shows the following items:

- SomeEnum aaa
- SomeEnum bbb
- Macros



GSoC Work: Macro navigation

- Show definition text and uses of Macro definitions

The screenshot shows the KDevelop IDE with a C++ file named `main.cpp` open. The code in the editor is as follows:

```
1 #define FOO_BAR(x) int x = 0; (void)x;
2 |
3 int main(int argc, char** argv)
4 {
5     FOO_BAR(a);
6     FOO_BAR(b);
7 }
8
```

A tooltip is displayed over the macro definition, showing the text: "Function macro: `FOO_BAR(x)`, defined in `main.cpp :1` [Show uses](#)". Below the tooltip, the macro's body is shown: `int x = 0; (void)x;`.

Below the editor, the "Uses of `FOO_BAR`" panel is visible. It shows "2 uses found" and lists the following:

- Declaration: `Line 1 #define FOO_BAR(x) int x = 0; (void)x;`
- In `Global`:
 - `Line 5 FOO_BAR(a);`
 - `Line 6 FOO_BAR(b);`



GSoC Work: Objective-C

- A little bit of Objective-C support (patches welcome!)

```
Car.m x Foundation.h x NSDebug.h x main.cpp x
1  #import <Foundation/Foundation.h>
2
3  @interface HelloWorldExample: NSObject
4  - (void) printMethod;
5  @end
6
7  @implementation HelloWorldExample
8  - (void) printMethod {
9  NSLog(@"Hello World \n");
10 }
11 @end
12
13 int main ()
14 {
15 /* Main method a starting point
16 in every Objective-C programs */
17 HelloWorldExample * foo = [[foo alloc] init];

```

class [HelloWorldExample](#) (resolved forward-declaration: [HelloWorldExample](#))
Kind: **Forward Declaration**
Decl.: [Car.m :3](#) [Show uses](#)



GSoC Work: More assistants

- Clang parses Doxygen-style comments!

```
1  /**
2   * @param bar The first parameter
3   */
4  int func(int foo)
5  {
6   ....
7  }
8
9
```

Line: 6 Col: 5

Documents

- ...make-t
- main.

Problem in **Semantic analysis**:
Parameter 'bar' not found in the function declaration [-Wdocumentation]

Hint: Did you mean 'foo'?

Solve: Fix-it Hints



GSoC Report

- TODOs
 - **General tasks:** Polish completion features (e.g. appending a "(" when completing a function call)
 - **Qt integration:** Support completion inside `QObject::connect` statements, `Q_PROPERTY` macros, ...
 - **Refactoring tools:** Investigate Clang refactoring tools (libtooling)
 - **Windows support:** Test `kdev-clang` on Windows (Clang front-end itself works just fine on Windows)

We're working on it!



Thanks!



References

- **Introduction to libclang:**
http://clang.llvm.org/clang_video-05-25-2007.html
- **C interface to Clang:**
http://clang.llvm.org/doxygen/group__CINDEX.html
- **GSoC: Clang Integration in KDevelop:**
<http://kfunk.org/2014/04/28/gsoc-2014-improving-the-clang-integration-in-kdevelop/>
(+ follow-up posts)
- **Try it out yourself:**
<http://milianw.de/blog/katekdevelop-sprint-2014-let-there-be-clang>
[Section: Take my Code]