Clang.jl Documentation

Release 0.1

Isaiah H. Norton

December 28, 2015

Contents

CHAPTER 1

Index API

parse_header (*header::String; [index, diagnostics, cplusplus, clang_args, clang_includes, clang_flags]*)

Main entry-point to Clang.jl. The required `header` argument specifies a header to parse. Returns the top CLCursor in the resulting TranslationUnit. Optional (keyword) arguments are as follows:

Parameters

- **index** (*Ptr{CXIndex}*) – Pass to re-use a CXIndex over multiple runs.
- **diagnostics** (*Bool*) – Print Clang diagnostics to STDERR.
- **cplusplus** (*Bool*) – Parse as C++ file.
- **args** (*Vector{String}*) – Vector of arguments (strings) to pass to Clang.
- **includes** (*Vector{String}*) – Vector of include paths for Clang to search (note: path only, "-I" will be prepended automatically)
- **flags** – Bitwise OR of TranslationUnit_FLags enum. Not required in typical use; see libclang manual for more information.

Return type TranslationUnit (<: CLCursor)

children (*c::CLCursor*)

Retrieve child nodes for given CLCursor. Julia's iterator protocol is supported, allowing constructs such as:

```
for node in children(cursor)
    ...
end
```

cu_type (*c::CLCursor*)

Get associated CLType for a given CLCursor.

return_type (*c::{FunctionDecl, CXXMethod}*)

Get the CLType returned by the function or method.

name (*c::CLCursor*)

Return the display name of a given CLCursor. This is the "long name", and for a FunctionDecl will be the full function call signature (function name and argument types).

spelling (*t::CLCursor*)
spelling (*t::CLType*)

Return the spelling of a given CLCursor or CLType. Spelling is the "short name" of a given element. For a FunctionDecl the spelling will be the function name only (similarly the identifier name for a RecordDecl or TypedefDecl cursor).

value (*c::EnumConstantDecl*)

Returns the value of a given EnumConstantDecl, automatically using correct call for signed vs. unsigned types.

pointee_type(*t::Pointer*)
Returns the type of the element pointed to by a Pointer cursor.

CHAPTER 2

Types

type `CLType`

Datatype representation.

Intrinsic datatypes	
VoidType	
BoolType	
Char_U	
UChar	
Char16	
Char32	
UShort	
UInt	
ULong	
ULongLong	
UInt128	
Char_S	
SChar	
WChar	
Short	
IntType	
Long	
LongLong	
Int128	
Float	
Double	
LongDouble	
NullPtr	

Other useful types	
Invalid	
Unexposed	
Record	
Pointer	
Typedef	
Enum	
Vector	
ConstantArray	
Overload	
Dependent	
FirstBuiltin	
LastBuiltin	
Complex	
BlockPointer	
LValueReference	
RValueReference	
FunctionNoProto	
FunctionProto	

type `CLCursor`

AST node types:

UnexposedDecl	
StructDecl	
UnionDecl	
ClassDecl	
EnumDecl	
FieldDecl	
EnumConstantDecl	
FunctionDecl	
VarDecl	
ParmDecl	
TypedefDecl	
CXXMethod	
Namespace	
LinkageSpec	
Constructor	
Destructor	
ConversionFunction	
TemplateTypeParameter	
NonTypeTemplateParameter	
TemplateTemplateParameter	
FunctionTemplate	
ClassTemplate	
ClassTemplatePartialSpecialization	
NamespaceAlias	
UsingDirective	
UsingDeclaration	
TypeAliasDecl	
CXXAccessSpecifier	
FirstDecl	

Continued on next page

Table 2.1 – continued from previous page

LastDecl	
FirstRef	
TypeRef	
CXXBaseSpecifier	
TemplateRef	
NamespaceRef	
MemberRef	
LabelRef	
OverloadedDeclRef	
VariableRef	
LastRef	
FirstInvalid	
InvalidFile	
NoDeclFound	
NotImplemented	
InvalidCode	
LastInvalid	
FirstExpr	
UnexposedExpr	
DeclRefExpr	
MemberRefExpr	
CallExpr	
BlockExpr	
IntegerLiteral	
FloatingLiteral	
ImaginaryLiteral	
StringLiteral	
CharacterLiteral	
ParenExpr	
UnaryOperator	
ArraySubscriptExpr	
BinaryOperator	
CompoundAssignOperator	
ConditionalOperator	
CStyleCastExpr	
CompoundLiteralExpr	
InitListExpr	
AddrLabelExpr	
StmtExpr	
GenericSelectionExpr	
GNUNullExpr	
CXXStaticCastExpr	
CXXDynamicCastExpr	
CXXReinterpretCastExpr	
CXXConstCastExpr	
CXXFunctionalCastExpr	
CXXTypeidExpr	
CXXBoolLiteralExpr	
CXXNullPtrLiteralExpr	
CXXThisExpr	

Continued on next page

Table 2.1 – continued from previous page

CXXThrowExpr	
CXXNewExpr	
CXXDeleteExpr	
UnaryExpr	
PackExpansionExpr	
SizeOfPackExpr	
LambdaExpr	
LastExpr	
FirstStmt	
UnexposedStmt	
LabelStmt	
CompoundStmt	
CaseStmt	
DefaultStmt	
IfStmt	
SwitchStmt	
WhileStmt	
DoStmt	
ForStmt	
GotoStmt	
IndirectGotoStmt	
ContinueStmt	
BreakStmt	
ReturnStmt	
GCCAsmStmt	
AsmStmt	
CXXCatchStmt	
CXXTryStmt	
CXXForRangeStmt	
SEHTryStmt	
SEHExceptStmt	
SEHFinallyStmt	
MSAsmStmt	
NullStmt	
DeclStmt	
LastStmt	
TranslationUnit	
FirstAttr	
UnexposedAttr	
IBActionAttr	
IBOutletAttr	
IBOutletCollectionAttr	
CXXFinalAttr	
CXXOverrideAttr	
AnnotateAttr	
AsmLabelAttr	
LastAttr	
PreprocessingDirective	
MacroDefinition	
MacroExpansion	

Continued on next page

Table 2.1 – continued from previous page

MacroInstantiation	
InclusionDirective	
FirstPreprocessing	
LastPreprocessing	
ModuleImportDecl	
FirstExtraDecl	
LastExtraDecl	

CHAPTER 3

C Wrapper Generator

The Clang.jl wrapper generator is designed to be simple but flexible. The most basic invocation looks like this:

```
context = wrap_c.init()

headers = ["all.h", "your.h", "headers.h"]
wrap_c.wrap_c_headers(context, headers)
```

However, it is usually necessary to set compiler arguments or customize output. The `init` function provides several arguments to configure the compiler, as well as callback functions used to determine various aspects of the output.

init (*; index*, *output_file*, *clang_args*, *clang_includes*, *clang_diagnostics*, *header_wrapped*, *header_library*, *header_outputfile*)

`init`: Create wrapping context. Keyword args are available to specify options, but all options are given sane defaults.

Parameters

- **index** – Union{}
- **output_file** (*ASCIIString*) –
- **common_file** (*ASCIIString*) – Name of common output file (types, constants, typealiases)
- **clang_args** (*Vector{String}*) –
- **clang_includes** (*Vector{String}*) – List of include paths for Clang to search.
- **clang_diagnostics** (*Bool*) – Display Clang diagnostics
- **header_wrapped** (*Function(header_file::ASCIIString, cursor_name::ASCIIString) -> Bool*) – Function called to determine whether a header should be wrapped.
- **header_library** – Function called to determine the library name for a given header.
- **header_outputfile** (*Function(header::ASCIIString) -> Bool*) – Function called to determine the output filename for a given header.
- **rewriter** (*Function(Expr)*) – Function to rewrite generated expressions

wrap_c_headers (*wc::WrapContext*, *headers::Vector{String}*)

Generate a wrapping using given WrapContext for a list of headers. All parameters are governed by the WrapContext, see `wrap_c.init` for full listing of options.

type WrapContext

`WrapContext` stores all information about the wrapping job.

Field index CXIndex to reuse for parsing.

Field output_file Name of main output file.

Field common_file Name of common output file (constants, types, and aliases)

Field clang_includes Clang include paths

Field clang_args additional {"-Arg", "value"} pairs for clang

Field header_wrapped called to determine cursor inclusion status

Field header_library called to determine shared library for given header

Field header_outfile called to determine output file group for given header

Field common_buf Array

Field cache_wrapped [Internal] Set{ASCIIString}

Field output_bufs [Internal] DefaultOrderedDict{ASCIIString, Array{Any}}

Field options InternalOptions

CHAPTER 4

Debugging

4.1 General

- pass *diagnostics = true* to *cindex.parse_header*
- set *clang_args = ["-v"]* in *cindex.parse_header*

4.2 Missing AST Items

If some AST elements (for example, FunctionDecl entries) appear to be missing, verify that all headers are locatable. To diagnose, pass *diagnostics = true* to *cindex.parse_header*. Doing so may uncover the following error.

4.3 Missing stddef.h (or others)

See *http://clang.llvm.org/docs/LibTooling.html#libtooling-builtin-includes*

C

I

N

P

R

S

V

W