

What to Expect from a Next-Generation C++ Build System

Boris Kolpackov

Code Synthesis

v1.0, September 2018

***CODE
SYNTHESIS***

Change is in the Air

Our “*git moment*”

Why Now?

What's changing?

- C++ Modules
- Packaging & Dependency Management
- Distributed Compilation and Caching
- C++ Std is Paying Attention (SG15)

Which One?

*“I am happy with any build system
as long as it's the one I use.”*

Values

What are C++ Community Core Values?

“Platform as Reflection of Values”

Values

Approachability	Integrity	Robustness
Availability	Maintainability	Safety
Compatibility	Measurability	Security
Composability	Operability	Simplicity
Debuggability	Performance	Stability
Expressiveness	Portability	Thoroughness
Extensibility	Resiliency	Transparency
Interoperability	Rigor	Velocity

C Values

*C is a **general-purpose** programming language which features **economy** of expression, modern control flow and data structures, and a rich set of operators. C is **not a “very high level”** language, **nor a “big”** one, and is not specialized to any particular area of application. But its absence of restrictions and its generality make it more convenient and effective for many tasks than supposedly more powerful languages.*

The C Programming Language

C Values

Approachability	Integrity	Robustness
Availability	Maintainability	Safety
Compatibility	Measurability	Security
Composability	Operability	Simplicity
Debuggability	Performance	Stability
Expressiveness	Portability	Thoroughness
Extensibility	Resiliency	Transparency
Interoperability	Rigor	Velocity

C++ Values

*C++ is a **general-purpose** programming language designed to make programming more **enjoyable** for the **serious** programmer. Except for minor details, C++ is a **superset** of the C programming language. In addition to the facilities provided by C, C++ provides **flexible and efficient** facilities for defining new types.*

The C++ Programming Language, 1st ed

C++ Values

Approachability	Integrity	Robustness
Availability	Maintainability	Safety
Compatibility	Measurability	Security
Composability	Operability	Simplicity
Debuggability	Performance	Stability
Expressiveness	Portability	Thoroughness
Extensibility	Resiliency	Transparency
Interoperability	Rigor	Velocity

C++11 Values

C++ is a **general-purpose** programming language emphasizing the design and use of **type-rich**, **lightweight** abstractions. It is particularly suited for **resource-constrained** applications, such as those found in software infrastructures. C++ rewards the programmer who takes the time to master techniques for writing **quality** code. C++ is a language for someone who takes the task of programming **seriously**.

The C++ Programming Language, 4th ed

C++11 Values

Approachability	Integrity	Robustness
Availability	Maintainability	Safety
Compatibility	Measurability	Security
Composability	Operability	Simplicity
Debuggability	Performance	Stability
Expressiveness	Portability	Thoroughness
Extensibility	Resiliency	Transparency
Interoperability	Rigor	Velocity

C++11 Values

Approachability	Integrity	Robustness
Availability	Maintainability	Safety
Compatibility	Measurability	Security
Composability	Operability	Simplicity
Debuggability	Performance	Stability
Expressiveness	Portability	Thoroughness
Extensibility	Resiliency	Transparency
Interoperability	Rigor	Velocity

C++11 Values

Approachability	Integrity	Robustness
Availability	Maintainability	Safety
Compatibility	Measurability	Security
Composability	Operability	Simplicity
Debuggability	Performance	Stability
Expressiveness	Portability	Thoroughness
Extensibility	Resiliency	Transparency
Interoperability	Rigor	Velocity

C++11 Values

Approachability

Integrity

Robustness

Availability

Maintainability

Safety

Compatibility

Measurability

Security

Composability

Operability

Simplicity

Debuggability

Performance

Stability

Expressiveness

Portability

Thoroughness

Extensibility

Resiliency

Transparency

Interoperability

Rigor

Velocity

Modern C++ Values

- Performance
- Extensibility & Expressiveness
- Portability & Compatibility
- Robustness
- Approachability

Modern C++ Values

- Performance
- Extensibility & Expressiveness == *Unconstrained*
- Portability & Compatibility
- Robustness
- Approachability

JavaScript Values

Approachability

Integrity

Robustness

Availability

Maintainability

Safety

Compatibility

Measurability

Security

Composability

Operability

Simplicity

Debuggability

Performance

Stability

Expressiveness

Portability

Thoroughness

Extensibility

Resiliency

Transparency

Interoperability

Rigor

Velocity

C++ Values

- Performance
- Extensibility & Expressiveness
- Portability & Compatibility
- Robustness
- Approachability

Build Systems

Meta-Questions and Overall Design

Build System-Less?

“The best build system is the one you don’t need.”

Native or Meta (Project Generator)?

Meta Build System

*Race to the bottom,
to the lowest common denominator.*

- C++ Modules
- Generated Source Code
- Distributed Compilation/Caching
- Compilation Database

Meta Build System

What about IDE support?

Tail wagging the dog?

Native Build System

Uniform: works the same everywhere,
no project generation step.

Meta vs Native: Values

Meta	Native
Performance	Performance
Extensibility & Expressiveness	Extensibility & Expressiveness
Portability & Compatibility	Portability & Compatibility
Robustness	Robustness
Approachability	Approachability

Meta would be a Fundamental Mistake

Black Box or a Concept of Build?

Black Boxes and Magic

```
# make
```

```
#
```

```
hello$(EXE): hello.$(OBJ)  
    $(CXX) -o $@ $^
```

```
hello.$(OBJ): hello.cxx
```

```
%.${OBJ}: %.cxx  
    $(CXX) -o $@ -c $<
```

```
# CMake
```

```
#
```

```
project (hello)  
add_executable(hello hello.cxx)
```

Black Box vs Build Model: Values

Black Box	Build Model
Performance	Performance
Extensibility & Expressiveness	Extensibility & Expressiveness
Portability & Compatibility	Portability & Compatibility
Robustness	Robustness
Approachability	Approachability

Implementation Language

Should we depend on another “system”?

Is C++ inadequate for the task?

Implementation Language: Values

Java/Python/...

C++

Performance

Performance

Extensibility & Expressiveness

Extensibility & Expressiveness

Portability & Compatibility

Portability & Compatibility

Robustness

Robustness

Approachability

Approachability

Declarative or Scripted?

problem warrants a purpose-built language

Declarative vs Scripted: Values

Scripted

Performance

Extensibility & Expressiveness

Portability & Compatibility

Robustness

Approachability

Declarative

Performance

Extensibility & Expressiveness

Portability & Compatibility

Robustness

Approachability

Declarative vs Scripted

Hybrid, Mostly Declarative?

- Typed variables.
- Pure functions.
- Exclusions (`if-else`).
- Repettions (`for-loop`).
- Custom functions and rules.

Next-Generation C++ Build System

- Native
- Has a conceptual model of build
- Implemented (and extensible) in C++
- With mostly declarative, type-safe build language

Next-Generation C++ Build System

- Native
- Has a conceptual model of build
- Implemented (and extensible) in C++
- With mostly declarative, type-safe build language
- Part of the dependency management toolchain

Next-Generation C++ Build System

- Native
- Has a conceptual model of build
- Implemented (and extensible) in C++
- With mostly declarative, type-safe build language
- Part of the dependency management toolchain
- Available as a library for easier IDE/tools integration

build2 Values

build2 is a native, cross-platform build system with a terse, mostly declarative description language, a conceptual model of build, and a uniform interface with consistent behavior across platforms and compilers. build2 is an “honest” build system without magic or black boxes. You can expect to understand what’s going on underneath and be able to customize most of its behavior to suit your needs.

The build2 Build System

Black Boxes and Magic

make

#

```
hello$(EXE): hello.$(OBJ)
    $(CXX) -o $@ $^
```

```
hello.$(OBJ): hello.cxx
```

```
%. $(OBJ): %.cxx
    $(CXX) -o $@ -c $<
```

CMake

#

```
add_executable(hello hello.cxx)
```

build2

#

```
exe{hello}: cxx{hello}
```

Build Systems

Current-Generation Functionality

In/Out of Source Builds

Wildcard Patterns

“build-system-less” model for simple projects

Cross-Compilation is the Norm

(and not an afterthought)

Cross-Testing

Additional Operations:

- test
- install/uninstall (with pkg-config support)
- dist (preparation of source distributions)
- configure

Integrated Configuration Management

Build Systems

Next-Generation Functionality

Project Composability

- Importation
- Suprojects/Amalgamations (bundling)
- Import Installed

Scripted Testing

- Concise
- Portable
- Input generation
- Output analysis (regex)
- Parallel execution
- Incremental testing

High-Fidelity Hermetic Builds

detect/prevent changes to:

- Environment
- Tools (compilers, linkers, etc)
- Options
- Source code sets

Precise Change Detection

avoid recompiling of *ignorable* changes

C++ Modules

- How to discover imported modules
- How to map module names to file names
- Provide feedback to WG21 ([P1052R0](#), [P1156R0](#))
- “[Building C++ Modules](#)” (CppCon 2017)

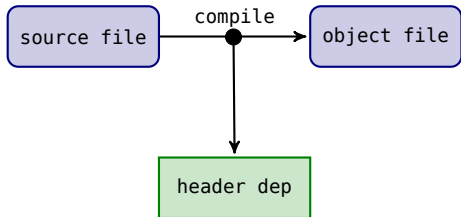
Distributed Compilation and Caching

reliable and generally-available

New C++ Build Model

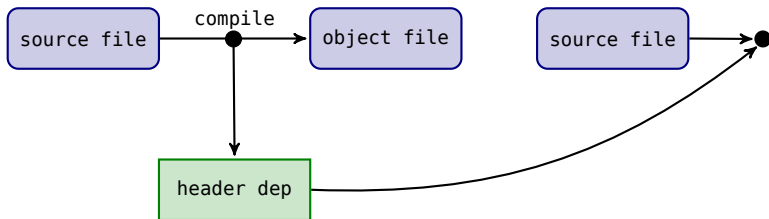
- C++ Modules
- Auto-generated headers
- Distributed compilation
- Ignorable change detection

Old C++ Build Model



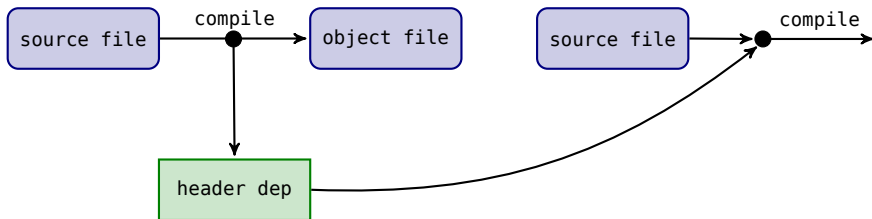
header dependency as byproduct of compilation

Old C++ Build Model



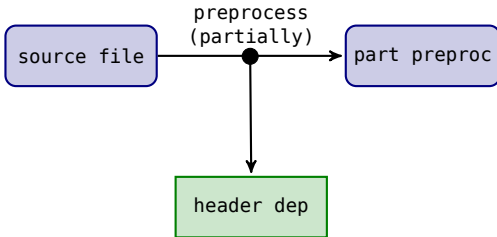
header dependency as byproduct of compilation

Old C++ Build Model

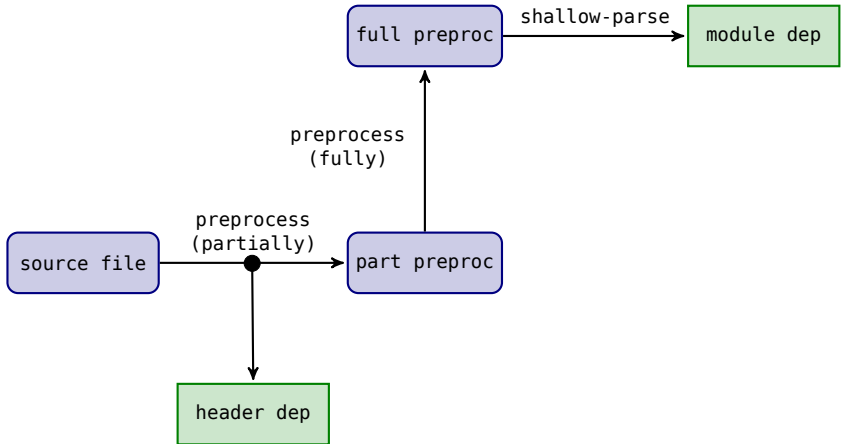


header dependency as byproduct of compilation

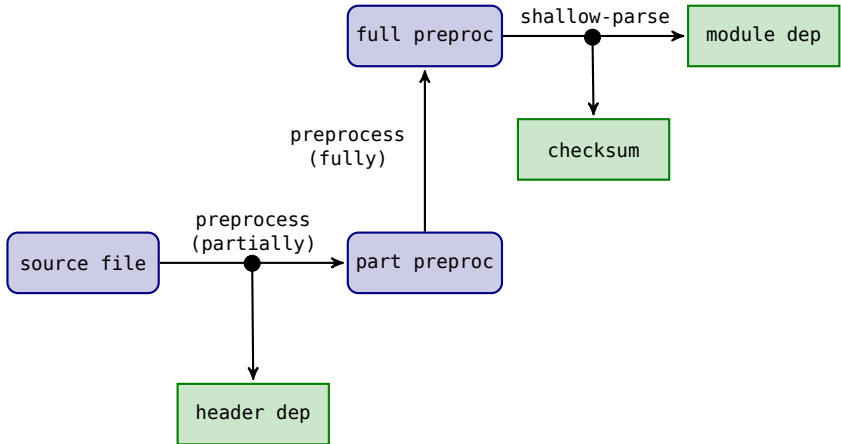
New C++ Build Model



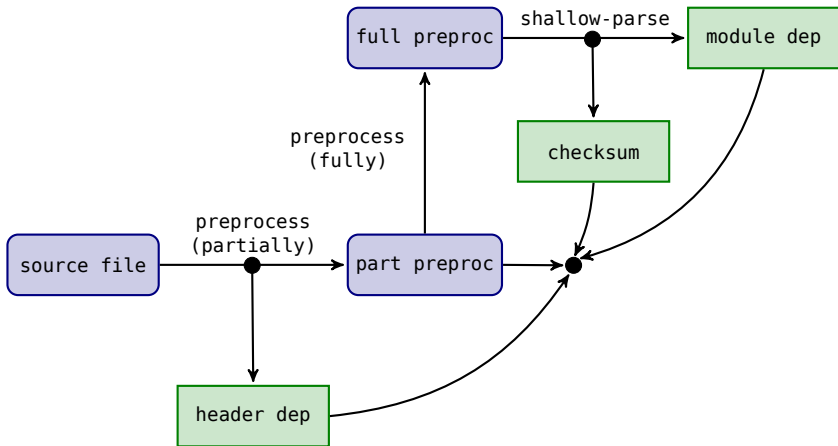
New C++ Build Model



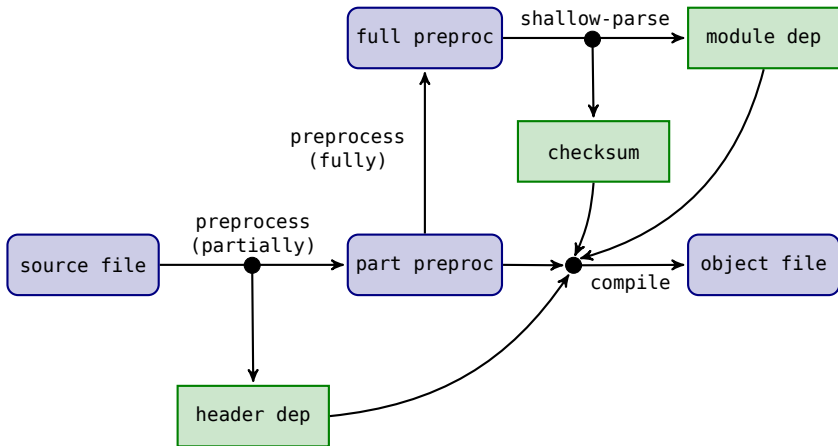
New C++ Build Model



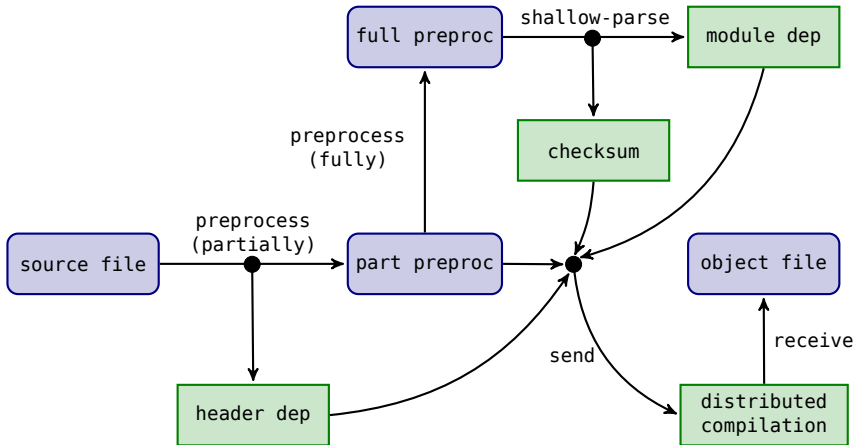
New C++ Build Model



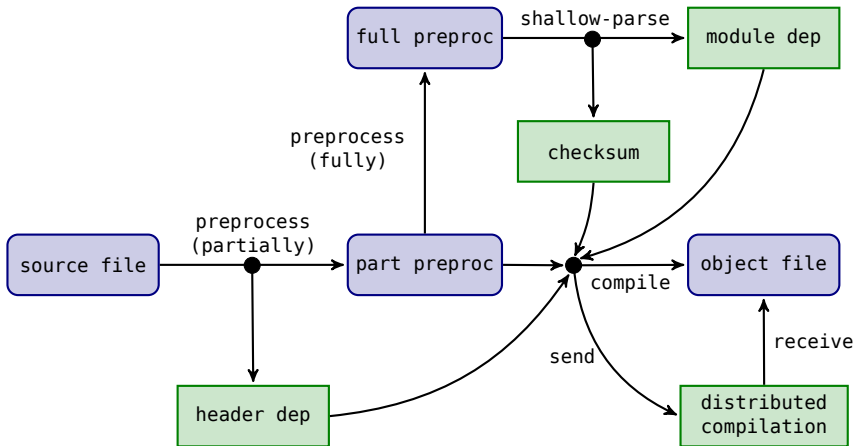
New C++ Build Model



New C++ Build Model



New C++ Build Model



Next-Generation C++ Build System

- Native
- Conceptual model of build
- Implemented/Extensible in C++
- Mostly declarative language
- Part of dependency management toolchain