# The Design of a New C++ Build Toolchain

Boris Kolpackov

Code Synthesis

v1.3, May 2016

## Extra Material

Terminal: `terminal.txt`

Example: `example.tar.gz`

## Who is this Guy?

- ODB, XSD
- GNU `make`-based build system, called `build`
- `build2` toolchain

## Who is this Guy?

“New Build System for New C++”

C++Now 2015

## What’s the Goal?

What is “better”?

## What’s the Goal?

Uniform Build Interface Across Platforms/Compilers

```
$ b config.cxx=g++-5 hello/
c++ cxx{hello}
ld exe{hello}
```

```
C:\> b config.cxx=cl.exe hello\
c++ cxx{hello}
ld exe{hello}
```

## What’s the Goal?

Ready Out of the Box, even on Windows

- No MinGW/Cygwin
- No Python
- No Linux userland

## What’s the Goal?

Development & Distribution

## What’s the Goal?

Reliable Builds

## What’s the Goal?

Cross Compilation

## What's the Goal?

Source Code Generation

## What’s the Goal?

No Black Boxes

## What’s the Goal?

Sane Syntax

## The build2 Toolchain

- `b` – build system driver
- `bpkg` – package manager
- `brep` – repository web interface
- `https://cppget.org`

## The build2 Toolchain

- Open source (MIT)
- Written in C++11 (GCC 4.8, Clang 3.4)
- Self-hosted and self-packaged
- Linux, Mac OS X, FreeBSD (Windows coming)

## The build2 Toolchain

Time for Some Examples?

## TERMINAL

`repo-web-interface`

`package-manager-usage`

`examine-manifest`

`build-system-usage`

`persistent-configuration`

`out-of-source-builds`

## Names

```
# buildfile
#
import libs = libhello%lib{hello}

exe{hello}: cxx{hello} $libs
```

## Names

```
# buildfile
#
import libs = libhello%lib{hello}

exe{hello}: cxx{hello} $libs
```

```
# makefile
#
hello: hello.cxx -lhello
```

## Names

```
# makefile
#
ifdef WINDOWS
  EXT := .exe
else
  EXT :=
endif

hello$EXT: hello.cxx -lhello
```

## Names

```
[proj%]type{name}
```

## Names

```
[proj%]type{name}
```

```
# buildfile
#
import libs = libhello%lib{hello}

exe{hello}: cxx{hello} $libs
```

## Name Generation

```
lib{butl}:                                \
{hxx         cxx}{ base64           } \
{hxx         cxx}{ char-scanner } \
{hxx         cxx}{ fdstream        } \
{hxx ixx     cxx}{ filesystem    } \
{hxx            }{ optional        } \
{hxx         cxx}{ pager            } \
{hxx ixx txx cxx}{ path              } \
{hxx            }{ path-map        } \
{hxx     txx    }{ prefix-map    } \
{hxx ixx     cxx}{ process          } \
{hxx         cxx}{ sha256           } \
{hxx     txx    }{ string-table } \
{hxx         cxx}{ timestamp      } \
{hxx         cxx}{ triplet          } \
{hxx            }{ utility          } \
{hxx            }{ vector-view   }
```

## Relative vs Absolute

```
# makefile
#
hello: hello.cxx -lhello
```

## Relative vs Absolute

```
# makefile
#
hello: hello.cxx -lhello
```

- UNIX-only, no naming variations

## Relative vs Absolute

```
# makefile
#
hello: hello.cxx -lhello
```

- UNIX-only, no naming variations
- Simple utilities, no sub-directories

## Relative vs Absolute

```
# makefile
#
hello: hello.cxx -lhello
```

- UNIX-only, no naming variations
- Simple utilities, no sub-directories
- `out == src`

## Relative vs Absolute

```
# makefile
#
hello: hello.cxx -lhello
```

- UNIX-only, no naming variations
- Simple utilities, no sub-directories
- `out == src`

- Path are relative to `CWD`

## Relative vs Absolute

```
# makefile
#
hello: hello.cxx -lhello
```

- UNIX-only, no naming variations
- Simple utilities, no sub-directories
- `out == src`

- Path are relative to `CWD`
- No `src/out` distinction or support

## Relative vs Absolute

```
# makefile
#
hello: hello.cxx -lhello
```

- UNIX-only, no naming variations
- Simple utilities, no sub-directories
- `out == src`

- Path are relative to `CWD`
- No `src/out` distinction or support
- Global variables

## Directory Scope

```
libhello/
└── hello/

libhello-gcc/
└── hello/
```

## Directory Scope

```
libhello/
└── hello/

libhello-gcc/
└── hello/


libhello/@libhello-gcc/
{
  libhello/hello/@libhello-gcc/hello/
  {

  }
}
```

## Directory Scope

```
libhello/
└── hello/

libhello-gcc/
└── hello/


libhello/@libhello-gcc/
{
  libhello/hello/@libhello-gcc/hello/
  {
    lib{hello}: {hxx cxx}{hello}
  }
}
```

## Scope Variables

```
libhello/@libhello-gcc/
{
  src_root = .../libhello/
  out_root = .../libhello-gcc/

  libhello/hello/@libhello-gcc/hello/
  {
    lib{hello}: {hxx cxx}{hello}

    cxx.poptions += -Isrc_root
  }
}
```

## Scope Variables

```
libhello/@libhello-gcc/
{
  src_root = .../libhello/
  out_root = .../libhello-gcc/

  libhello/hello/@libhello-gcc/hello/
  {
    lib{hello}: {hxx cxx}{hello}

    cxx.poptions += -Isrc_root
  }
}
```

## Scope Variables

```
libhello/@libhello-gcc/
{
  src_root = .../libhello/
  out_root = .../libhello-gcc/

  libhello/hello/@libhello-gcc/hello/
  {
    lib{hello}: {hxx cxx}{hello}

    cxx.poptions += -Isrc_root
  }
}
```

## TERMINAL

```
multi-config-build
```

## Variable Overrides

```
$ b config.cxx=clang++

$ b config.cxx.coptions+=-g

$ b config.cxx.poptions=+-I/tmp/fix
```

## TERMINAL

```
var-override
```

## High Fidelity Builds

Out-of-`buildfile` dependencies

## High Fidelity Builds

### Out-of-`buildfile` dependencies

- `#include`'ed headers (`-M*` / `.d`)

## High Fidelity Builds

### Out-of-`buildfile` dependencies

- `#include`’ed headers (`-M*` / `.d`)
- compiler: change (GCC to Clang), upgrade, reconfiguration

## High Fidelity Builds

### Out-of-`buildfile` dependencies

- `#include`'ed headers (`-M*` / `.d`)
- compiler: change (GCC to Clang), upgrade, reconfiguration
- options: `-O2` to `-O3`, add/remove `-I`

## High Fidelity Builds

### Out-of-`buildfile` dependencies

- `#include`'ed headers (`-M*` / `.d`)
- compiler: change (GCC to Clang), upgrade, reconfiguration
- options: `-O2` to `-O3`, add/remove `-I`
- input(s): `foo.cpp` to `foo.cxx`, (re)move file from `lib/exe`

## High Fidelity Builds

Store values/sha256 checksums in `.d` files

## High Fidelity Builds

Store values/sha256 checksums in `.d` files

“Auxiliary Dependency Database”

## TERMINAL

```
examine-dot-d
```

## TERMINAL

```
compiler-checksum
```

## TERMINAL

```
hfb-example
```

## TERMINAL

```
compiler-detection
```

## TERMINAL

```
cross-compilation
```

## Operations

- `configure/disfigure`
- `update/clean`
- `install/uninstall`
- `test`
- `dist`

## Operations

```
# makefile
#
hello: hello.cxx -lhello
```

## Operations

```
# makefile
#
hello: hello.cxx -lhello

.PHONY: test install clean

clean:
	...

test: hello
	...

install: hello
	...
```

## Operations

Why this:

```
$ make && make install
```

## Operations

Why this:

```
$ make && make install
```

And not just:

```
$ make install
```

## Operations

Why this:

```
$ make && make install
```

And not just:

```
$ make install
$ sudo make install
```

## Operations

Targets Not the Right Concept

## Operations

Targets Not the Right Concept

Operations On the Dependency Graph

## Operations

Targets Not the Right Concept

Operations On the Dependency Graph

- Operations: `update clean test install`

## Operations

Targets Not the Right Concept

Operations On the Dependency Graph

- Operations: `update clean test install`
- Meta-operations: `configure dist`

# Operations

## Targets Not the Right Concept

## Operations On the Dependency Graph

- Operations: `update clean test install`
- Meta-operations: `configure dist`
- Pre/Post-operations: `update-for-{install test}`

## TERMINAL

```
install-libhello
```

## Import

```
# hello buildfile
#
import libs = libhello%lib{hello}

exe{hello}: cxx{hello} $libs
```

## Import

```
# hello buildfile
#
import libs = libhello%lib{hello}

exe{hello}: cxx{hello} $libs


# libhello 1.1.X buildfile
#
import libs  = libformat%lib{format}
import libs += libprint%lib{print}

lib{hello}: {hxx cxx}{hello} $libs
```

## Import

### Import Search

1. `config.import.<project>`
2. rule-specific search
3. fallback search

## Import

Rule-Specific Search

## Import

### Rule-Specific Search

```
# makefile
#
hello: hello.cxx -lhello
```

## Import

### Rule-Specific Search

```
# makefile
#
hello: hello.cxx -lhello
```

Where is `-lhello` searched for?

## Import

### Rule-Specific Search

Where should `-lhello` be searched for?

## Import

### Rule-Specific Search

Where should `-lhello` be searched for?

The same place where the compiler would:

```
$ g++ hello.cxx -lhello
```

## TERMINAL

```
import-installed
```

## Import

1. `config.import.<project>`
2. rule-specific search
3. fallback search

What could the Fallback Search be?

## Subprojects & Amalgamation

- Drop a project (*subproject*) into another (*amalgamation*)

## Subprojects & Amalgamation

- Drop a project (*subproject*) into another (*amalgamation*)
- Subprojects inherit amalgamation’s configuration

## Subprojects & Amalgamation

### Import Search

1. `config.import.<project>`
2. rule-specific search
3. fallback search

## Subprojects & Amalgamation

### Import Search

0. subproject search
1. `config.import.<project>`
2. rule-specific search
3. fallback search

## TERMINAL

```
libhello-subproject
```

## Subprojects & Amalgamation

`bpkg` configuration is amalgamation

## Subprojects & Amalgamation

bpkg configuration is amalgamation

- Packages auto-magically inherit configuration

## Subprojects & Amalgamation

bpkg configuration is amalgamation

- Packages auto-magically inherit configuration
- Packages auto-magically resolve prerequisites

## TERMINAL

```
examine-bpkg-configuration
```

## Package Manager

`bpkg`

## Package Manager

`bpkg`

- Uses SQLite (via ODB)

## Package Manager

`bpkg`

- Uses SQLite (via ODB)
- Repository signing/authentication

## Package Manager

`bpkg`

- Uses SQLite (via ODB)
- Repository signing/authentication
- How to handle requirements (C++11-only, !Windows)?

## Package Manager

bpkg

- Uses SQLite (via ODB)
- Repository signing/authentication
- How to handle requirements (C++11-only, !Windows)?
- How to handle conditional/runtime dependencies?

## Web Interface

```
brep
```

## Web Interface

`brep`

- Apache2 module in C++11

## Web Interface

`brep`

- Apache2 module in C++11
- Uses PostgreSQL (via ODB)

## Web Interface

`brep`

- Apache2 module in C++11
- Uses PostgreSQL (via ODB)
- Package search

## C++ Package Repository

`cppget.org`

## C++ Package Repository

`cppget.org`

- `stable/testing/beta/alpha/queue` sections

## C++ Package Repository

`cppget.org`

- `stable/testing/beta/alpha/queue` sections
- Tracked in `git` repository

## C++ Package Repository

`cppget.org`

- `stable/testing/beta/alpha/queue` sections
- Tracked in `git` repository
- Policies (licenses, name disputes, micro-packages)

## C++ Package Repository

`cppget.org`

- `stable/testing/beta/alpha/queue` sections
- Tracked in `git` repository
- Policies (licenses, name disputes, micro-packages)
- Is this Boost 2.0?

## Build Bot

`bbot`

(coming soon)

## What’s Next?

- VC++/Windows
- Documentation
- Parallel builds
- External modules & Inline C++ recipes

## Questions?

`build2.org`