Using build2, C++ Build Toolchain

Boris Kolpackov

Code Synthesis

v1.3, September 2016

What is “better”?

Philosophy

Engineering Tool

The Goal

Uniform Build Interface

```
   $ b config.cxx=g++-5  hello/

C:\> b config.cxx=cl.exe hello\
```

The Goal

Out of the Box, even on Windows

- No Python
- No MinGW / MSYS / Cygwin
- No Linux userland
- Ok C++ compiler ;-)

The Goal

Development & Distribution

The Goal

Reliable Builds

The Goal

Cross Compilation

The Goal

Source Code Generation

The Goal

No Black Boxes

The Goal

Sane Syntax

```
FOR_AllSubDirs(Its_a_<go>: ${all_good});
```

The build2 Toolchain

- `b` – build system driver
- `bpkg` – package manager
- `brep` – repository web interface
- `bbot` – build robot (in development)
- `https://cppget.org`

The build2 Toolchain

- Open source, MIT
- Written in C++14: GCC 4.8, Clang 3.4, VC 14u2
- Self-hosted and self-packaged, on `cppget.org`
- Platforms: Linux, Windows, Mac OS X, FreeBSD
- Compilers: GCC, Clang, VC, Intel icc

The build2 Toolchain

Examples?

Hello Example

```
// file: hello.cpp

#include <iostream>

using namespace std;

int main (int argc, char* argv[])
{
  if (argc != 2)
  {
    cerr << ”usage: ” << argv[0] << ” <name>” << endl;
    return 1;
  }

  cout << ”Hello, ” << argv[1] << ”!” << endl;
}
```

Hello Example

```
hello0/
│
└── hello.cpp
```

Hello Example

```
hello0/
│
├── build/
│
└── hello.cpp
```

bootstrap.build

```
# file: build/bootstrap.build

project = hello0

using config
using install
```

Hello Example

```
hello0/
│
├── build/
│    │
│    └── bootstrap.build
│
└── hello.cpp
```

root.build

```
# file: build/root.build

cxx.std = 11

using cxx

hxx{*}: extension = hpp
cxx{*}: extension = cpp
```

Hello Example

```
hello0/
│
├── build/
│   │
│   ├── root.build
│   └── bootstrap.build
│
└── hello.cpp
```

buildfile

```
# file: buildfile

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

Hello Example

```
hello0/
│
├── build/
│   │
│   ├── root.build
│   └── bootstrap.build
│
├── hello.cpp
└── buildfile
```

High Fidelity Builds

Out-of-`buildfile` dependencies

High Fidelity Builds

Out-of-`buildfile` dependencies

- `#include`'ed headers

High Fidelity Builds

Out-of-`buildfile` dependencies

- `#include`’ed headers
- compiler: change (GCC to Clang), upgrade

High Fidelity Builds

Out-of-`buildfile` dependencies

- `#include`’ed headers
- compiler: change (GCC to Clang), upgrade
- options: `-g` to `-O3`, add/remove `-I`/`-L`

High Fidelity Builds

Out-of-`buildfile` dependencies

- `#include`'ed headers
- compiler: change (GCC to Clang), upgrade
- options: `-g` to `-O3`, add/remove `-I`/`-L`
- input(s): remove source file from `lib`/`exe`

High Fidelity Builds

Store values/sha256 in `.d` files

High Fidelity Builds

Store values/sha256 in `.d` files

“Auxiliary Dependency Database”

Variable Overrides

```
$ b config.cxx=g++-5

$ b config.cxx.coptions+=-g

$ b config.cxx.loptions=+-L/tmp/lib
```

Operations

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

Import

- Way to connect projects
- Rule-specific search (import installed)
- `pkg-config` integration

Subprojects & Amalgamation

- Drop a project (*subproject*) into another (*amalgamation*)
- Subprojects inherit amalgamation's configuration
- `bpkg` configuration is amalgamation

Library Dependency Export

- Like `pkg-config` but *inside* the build system
- Library dependencies: *interface* vs *implementation*
- *Underlinking* vs *Overlinking*

Library Versioning

- Platform-independent: `libfoo-1.2.so`, `libfoo-1.2.dll`
- Platform-specific: `libfoo.so.1.2` (coming soon)
- Handles `soname`, etc.

Library Rpath

- Default for non-installed builds (run tests, etc.)
- Optional for installed builds (`config.bin.rpath`)
- Limited emulation on Windows (non-installed)

What’s Next?

`build2`:

- Parallel builds
- External modules & Inline C++ recipes
- Test module improvements (`testscript`)
- Documentation

`bpkg`:

- Query system package managers (`rpm`, `dpkg`, `pkg-config`)
- Build-time dependencies

`bbot`:

- Coming soon

Questions?

build2.org