The Boost.Build System

Vladimir Prus
ghost@cs.msu.su

Moscow State University and CodeSourcery, Inc.

SYRCoSE 2009

The Boost C++ Libraries

- Collection of portable C++ libraries
- Started by members of C++ Standards Committee
- Many libraries planned for standardization

Portability and Diversity

- Libraries should work everywhere
- Libraries *are* used everywhere
- Developers are rarely multiplatform
- Users are often unexperienced

Requirements

1. "Write once, build everywhere"
2. Extensibility
3. Multiple variants

Multiple variants

- Between different parts of the project
- Between different builds
- Within one build

Build systems basics: GNU Make

Basic target

```
a.o: a.c
	g++ -o a.o -g a.c
```

More flexible target

```
a.$(OBJEXT): a.c
	g++ -o a.o $(CFLAGS) a.c
```

Step 0: Modern Build Systems

Generator functions

```
library(helper, helper.c)
```

- Focus on indent, not details
- No “automatic” portability

Step 1: Portable Build Parameters

```
library(helper, helper.c,
                optimization=space)
```

- Every generator functions accepts the same parameters.
- Addresses requirement 1 (“Write once, build everywhere”)

Step 2: Generator selection

- One `library` cannot handle all platforms.
- Introduce platform-specific generators, e.g. `msvc.link.dll` and `gcc.link.dll`
- The `library` function select and forwards.
- Addresses requirement 2 (Extensibility)

Step 3: Metatargets

- Imperative semantics makes multiple variants hard
- Generators should not create targets
- Introduce *metatargets*
- Addresses requirement 3 (Multivariant builds)

Outline of operation

- Buildfiles are parsed, metatargets are created
- Requested properties are passed to “top level” metatargets
- Generator is selected
- Generator constructs targets.

Important detail: Requirements

```
exe a : a.cpp helper : threading=multi
                       toolset=msvc:link=static ;
...
lib helper : helper.cpp : : : <include>helper ;
```

Important detail: Generator selection

Generator	Type	Required parameters
gcc.link.dll	LIB	toolset=gcc
gcc.link	EXE	toolset=gcc
msvc.link.dll	LIB	toolset=msvc

Important detail: Command templates

```
actions gcc.link.dll {
    g++ -shared $(OPTIONS) -o $(<) $(>)
}
flags gcc.link.dll OPTIONS
    : profiling=on : -pg ;
```

True portability

Users say Boost.Build is indeed portable.

Metatargets are hard

- Every metatarget can be constructed in several variants
- There's no "current" compiler or any other parameter
- There are both metatargets and "ordinary" targets.

Code-level extension

- Original focus on ease for end users
- Internal extension interfaces not properly designed
- Example: generators
  - Generator with one output — 4 lines of code.
  - Generator with dynamic outputs — a page of code.

User expectations

- Completely fresh design
- Users don't care
  - "Why `CXX variable is ignored?`"
  - "`Why the commands are not printed?`"
  - "`Why it does not stop on first error?`"

Summary

Now

- Highly portable build system
- Suitable for production use

Future

- Incremental improvements
- Python port
- IDE integration

Questions?

- Boost.Build website: http://boost.org/boost-build2
- Boost.Build mailing list: boost-build@lists.boost.org
- The author: ghost@cs.msu.su