New Build System for New C++

Boris Kolpackov

Code Synthesis

v1.3, May 2015

Who is this Guy?

Who is this Guy?

- Contributed to GNU make
- 10 years of non-recursive build system, called `build`
- Successfully used for ODB, XSD
- Generates autotools, VC++ projects

ODB Configurations

- 5 Databases
- 6 Compilers/Versions
- 2 C++ modes (C++98 & C++11)
- 2 Qt versions (4 and 5)

ODB Configurations

- 5 Databases
- 6 Compilers/Versions
- 2 C++ modes (C++98 & C++11)
- 2 Qt versions (4 and 5)

120 Configurations

What's the Problem?

What's the Problem?

- "Thick" makefiles

What's the Problem?

- "Thick" makefiles
- "Hairy" distributions

What's the Problem?

- "Thick" makefiles
- "Hairy" distributions
- Linux-only

What's the Goal?

What's the Goal?

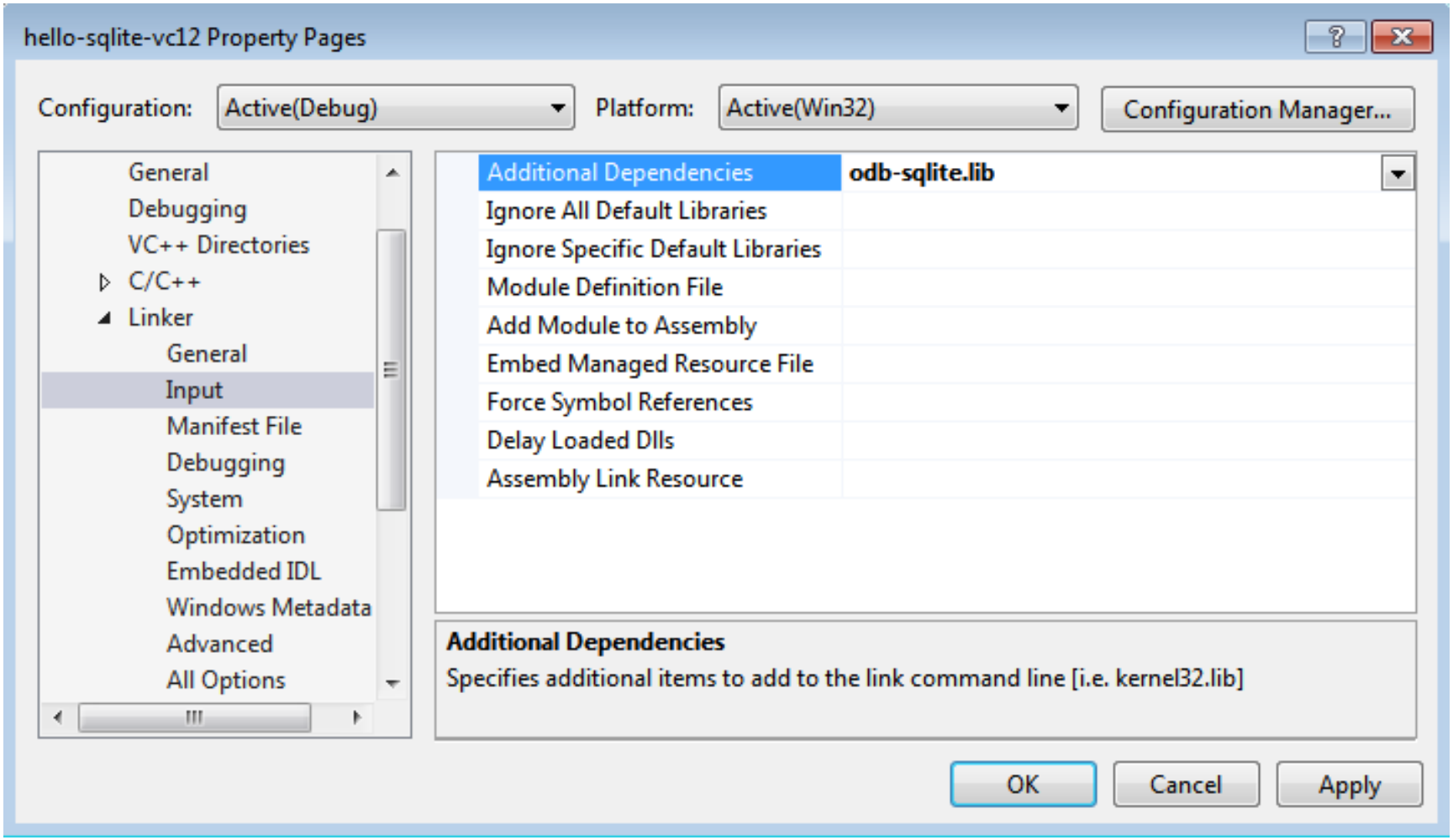
```
$ bget install --config gcc5-cxx14-debug libboost_datetime
```

What's the Goal?

```
$ bget install --config gcc5-cxx14-debug libboost_datetime

C:\> bget install --config vc12-x64-release libodb-sqlite
```

What's the Ultimate Goal?



How do We Get There?

How do We Get There?

"Those who don't understand make are condemned to reinvent it, poorly."

How do We Get There?

Let’s reinvent it properly!

How do We Get There?

If make is cvs...

How do We Get There?

If make is cvs...

Don’t you want git?

No Magic!

“Hello, World!”

```
$ ls
buildfile
hello.cxx
```

```
$ cat buildfile
```

```
using cxx

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

“Hello, World!” buildfile

```
using cxx

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

“Hello, World!” buildfile

```
using cxx

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

“Hello, World!” buildfile

```
using cxx

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

What is clean?

```
$ b clean
```

What is clean?

```
$ b clean
```

No mention of clean in our buildfile:

```
using cxx

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

What is clean?

Clean is an *operation*

What is clean?

Clean is an *operation*

Default operation is update

Still to be Explained

- Operations
- That "`test g++`" line
- That `dir{}` in "`dir{} already up to date`"

“Hello, World!” v2

```
$ ls
buildfile
hello.cxx
test.cxx
utility.cxx
utility.hxx
```

```
$ cat buildfile

using cxx
cxx.std = 11

exe{test}: cxx{test} cxx{utility}
exe{hello}: cxx{hello} cxx{utility}
```

“Hello, World!” v2 buildfile

```
using cxx
cxx.std = 11

exe{test}: cxx{test} cxx{utility}
exe{hello}: cxx{hello} cxx{utility}
```

“Hello, World!” v2 buildfile

```
using cxx
cxx.std = 11

exe{test}: cxx{test} cxx{utility}
exe{hello}: cxx{hello} cxx{utility}
```

“Hello, World!” v2 buildfile

```
using cxx
cxx.std = 11

exe{test}: cxx{test} cxx{utility}
exe{hello}: cxx{hello} cxx{utility}
```

Names

```
struct name
{
  string type;  // Optional.
  path dir;     // Optional.
  string value;
};
```

Names

```
struct name
{
  string type;  // Optional.
  path dir;     // Optional.
  string value;
};
```

```
dir/type{value}
```

Name Examples

```
foo bar                 # two simple names
{foo bar}               # same as above
exe{foo}                # name foo of type exe
exe{foo bar}            # two names, both exe
baz/foo                 # name foo in directory baz/
baz/{foo}               # same as above
baz/{foo bar}           # two names in baz/
baz/exe{foo bar}        # two names in baz/, both exe
baz/{foo exe{bar}}      # two names in baz/, last exe
exe{foo baz/{bar}}      # two names, last in baz/, both exe
```

Name Examples

```
foo bar                 # two simple names
{foo bar}               # same as above
exe{foo}                # name foo of type exe
exe{foo bar}            # two names, both exe
baz/foo                 # name foo in directory baz/
baz/{foo}               # same as above
baz/{foo bar}           # two names in baz/
baz/exe{foo bar}        # two names in baz/, both exe
baz/{foo exe{bar}}      # two names in baz/, last exe
exe{foo baz/{bar}}      # two names, last in baz/, both exe
```

Name Examples

```
foo bar                 # two simple names
{foo bar}               # same as above
exe{foo}                # name foo of type exe
exe{foo bar}            # two names, both exe
baz/foo                 # name foo in directory baz/
baz/{foo}               # same as above
baz/{foo bar}           # two names in baz/
baz/exe{foo bar}        # two names in baz/, both exe
baz/{foo exe{bar}}      # two names in baz/, last exe
exe{foo baz/{bar}}      # two names, last in baz/, both exe
```

Name Examples

```
foo bar                 # two simple names
{foo bar}               # same as above
exe{foo}                # name foo of type exe
exe{foo bar}            # two names, both exe
baz/foo                 # name foo in directory baz/
baz/{foo}               # same as above
baz/{foo bar}           # two names in baz/
baz/exe{foo bar}        # two names in baz/, both exe
baz/{foo exe{bar}}      # two names in baz/, last exe
exe{foo baz/{bar}}      # two names, last in baz/, both exe
```

Name Examples

```
foo bar                 # two simple names
{foo bar}               # same as above
exe{foo}                # name foo of type exe
exe{foo bar}            # two names, both exe
baz/foo                 # name foo in directory baz/
baz/{foo}               # same as above
baz/{foo bar}           # two names in baz/
baz/exe{foo bar}        # two names in baz/, both exe
baz/{foo exe{bar}}      # two names in baz/, last exe
exe{foo baz/{bar}}      # two names, last in baz/, both exe
```

Name Examples

```
foo bar                 # two simple names
{foo bar}               # same as above
exe{foo}                # name foo of type exe
exe{foo bar}            # two names, both exe
baz/foo                 # name foo in directory baz/
baz/{foo}               # same as above
baz/{foo bar}           # two names in baz/
baz/exe{foo bar}        # two names in baz/, both exe
baz/{foo exe{bar}}      # two names in baz/, last exe
exe{foo baz/{bar}}      # two names, last in baz/, both exe
```

Name Examples

```
foo bar                 # two simple names
{foo bar}               # same as above
exe{foo}                # name foo of type exe
exe{foo bar}            # two names, both exe
baz/foo                 # name foo in directory baz/
baz/{foo}               # same as above
baz/{foo bar}           # two names in baz/
baz/exe{foo bar}        # two names in baz/, both exe
baz/{foo exe{bar}}      # two names in baz/, last exe
exe{foo baz/{bar}}      # two names, last in baz/, both exe
```

Name Examples

```
foo bar                 # two simple names
{foo bar}               # same as above
exe{foo}                # name foo of type exe
exe{foo bar}            # two names, both exe
baz/foo                 # name foo in directory baz/
baz/{foo}               # same as above
baz/{foo bar}           # two names in baz/
baz/exe{foo bar}        # two names in baz/, both exe
baz/{foo exe{bar}}      # two names in baz/, last exe
exe{foo baz/{bar}}      # two names, last in baz/, both exe
```

Name Examples

```
foo bar                 # two simple names
{foo bar}               # same as above
exe{foo}                # name foo of type exe
exe{foo bar}            # two names, both exe
baz/foo                 # name foo in directory baz/
baz/{foo}               # same as above
baz/{foo bar}           # two names in baz/
baz/exe{foo bar}        # two names in baz/, both exe
baz/{foo exe{bar}}      # two names in baz/, last exe
exe{foo baz/{bar}}      # two names, last in baz/, both exe
```

Name Examples

```
foo bar                 # two simple names
{foo bar}               # same as above
exe{foo}                # name foo of type exe
exe{foo bar}            # two names, both exe
baz/foo                 # name foo in directory baz/
baz/{foo}               # same as above
baz/{foo bar}           # two names in baz/
baz/exe{foo bar}        # two names in baz/, both exe
baz/{foo exe{bar}}      # two names in baz/, last exe
exe{foo baz/{bar}}      # two names, last in baz/, both exe
```

Name Examples

```
foo bar                 # two simple names
{foo bar}               # same as above
exe{foo}                # name foo of type exe
exe{foo bar}            # two names, both exe
baz/foo                 # name foo in directory baz/
baz/{foo}               # same as above
baz/{foo bar}           # two names in baz/
baz/exe{foo bar}        # two names in baz/, both exe
baz/{foo exe{bar}}      # two names in baz/, last exe
exe{foo baz/{bar}}      # two names, last in baz/, both exe
```

“Hello, World!” v2 buildfile

```
using cxx
cxx.std = 11

#exe{test}: cxx{test} cxx{utility}
#exe{hello}: cxx{hello} cxx{utility}

exe{test}: cxx{test utility}
exe{hello}: cxx{hello utility}
```

The Default Target

What are we building here?

The Default Target

What are we building here?

We are building *this directory*

The Default Target

The default target is the current directory

The Default Target

The default target is the current directory

If not explicitly defined, then added implicitly with first target as prerequisite

Still to be Explained

- That `dir{}` in “`dir{}` already up to date”

Directory Target

```
dir{foo}
foo/dir{}
foo/
```

```
dir{}
dir{.}
./dir{}
./
.
```

Directory Target

```
dir{foo}
foo/dir{}
foo/
```

```
dir{}
dir{.}
./dir{}
./
.
```

Directory Target

```
dir{foo}
foo/dir{}
foo/
```

```
dir{}
dir{.}
./dir{}
./
.
```

“Hello, World!” v2 buildfile

```
using cxx
cxx.std = 11

exe{test}: cxx{test utility}
exe{hello}: cxx{hello utility}

.: exe{test hello}
```

“Hello, World!” v2 buildfile

```
using cxx
cxx.std = 11

exe{test}: cxx{test utility}
exe{hello}: cxx{hello utility}

.: exe{test hello}
```

Still to be Explained

- Operations
- That “`test g++`” line

Projects

Simple projects:

- Single `buildfile`
- No out-of-tree builds

Projects

Simple projects:

- Single `buildfile`
- No out-of-tree builds

“Real” projects:

- Sub-directories/multiple `buildfiles`
- Out-of-tree builds
- Project-wide settings
- Project import/export
- Subprojects/amalgamations

“Hello, World!” v3

```
hello/
├── hello/
│   ├── hello.cxx
│   ├── utility.cxx
│   └── utility.hxx
└── test/
    └── test.cxx
```

Project Roots and Bases

```
hello/                  <-- root
├── hello/              <-- base
│   ├── hello.cxx
│   ├── utility.cxx
│   └── utility.hxx
└── test/               <-- base
    └── test.cxx
```

Project Roots and Bases

```
hello/                  <-- src_root
├── hello/              <-- src_base
│   ├── hello.cxx
│   ├── utility.cxx
│   └── utility.hxx
└── test/               <-- src_base
    └── test.cxx

hello-debug/            <-- out_root
├── hello/              <-- out_base
│   ├── hello
│   ├── hello.o
│   ├── utility.o
└── test/               <-- out_base
    ├── test
    └── test.o
```

Project Roots and Bases

```
hello/
└── test/
    └── test.cxx

hello-debug/
└── test/
    ├── test
    └── test.o

src_root = /home/boris/hello/
src_base = /home/boris/hello/test/

out_root = /tmp/hello-debug/
out_base = /tmp/hello-debug/test/
```

“Hello, World!” v3

```
hello/
├── build/
│   ├── bootstrap.build
│   └── root.build
├── hello/
│   ├── hello.cxx
 ...
```

“Hello, World!” v3

```
hello/
├── build/
│   ├── bootstrap.build
│   └── root.build
├── hello/
│   ├── hello.cxx
 ...
```

“Hello, World!” v3

```
hello/
├── build/
│   ├── bootstrap.build
│   └── root.build
├── hello/
│   ├── hello.cxx
 ...
```

“Hello, World!” v3

```
hello/
├── build/
│   ├── bootstrap.build
│   └── root.build
├── hello/
│   ├── hello.cxx
 ...
```

bootstrap.build and root.build

```
# bootstrap.build
#
project = hello
```

bootstrap.build and root.build

```
# bootstrap.build
#
project = hello


# root.build
#
using cxx
cxx.std = 11
```

hello/buildfile

```
hello/
└── hello/
    └── buildfile
```

```
# hello/buildfile
#
exe{hello}: cxx{hello utility}
```

test/buildfile

```
hello/
└── test/
    └── buildfile
```

```
# test/buildfile
#
exe{test}: cxx{test ../hello/utility}
```

test/test.cxx

```
// test/test.cxx
//
#include ”utility.hxx”

...
```

test/test.cxx

```
// test/test.cxx
//
#include ”utility.hxx”

...
```

Include Issue

```
// test/test.cxx
//
#include ”../hello/utility.hxx”
```

Include Issue

```
// test/test.cxx
//
#include ”../hello/utility.hxx”


# test/buildfile
#
cxx.poptions += -I../hello
```

Include Fix

```
// test/test.cxx
//
#include <hello/utility.hxx>
```

Include Fix

```
// test/test.cxx
//
#include <hello/utility.hxx>


# root.build
#
using cxx
cxx.std = 11
cxx.poptions += -I$src_root
```

Root buildfile

```
hello/
└── buildfile
```

Root buildfile

```
hello/
└── buildfile
```

```
# buildfile
#
DO_SENSIBLE_THING_FOR_AllSubDirs(Its_a_go ${all_good});
```

Root buildfile

```
hello/
└── buildfile
```

```
# buildfile
#
d = hello/ test/
.: $d
include $d
```

Root buildfile

```
hello/
└── buildfile
```

```
# buildfile
#
d = hello/ test/
.: $d
include $d
```

Root buildfile

```
hello/
└── buildfile
```

```
# buildfile
#
d = hello/ test/
.: $d
include $d
```

Root buildfile

```
hello/
└── buildfile
```

```
# buildfile
#
d = hello/ test/
.: $d
include $d
```

Include

```
# buildfile
#
d = hello/ test/
.: $d
include $d
```

Include

```
# buildfile
#
d = hello/ test/
.: $d
include $d
```

Include

```
# buildfile
#
d = hello/ test/
.: $d

# hello/buildfile
#
exe{hello}: cxx{hello utility}

# test/buildfile
#
exe{test}: cxx{test ../hello/utility}
```

Include

```
# buildfile
#
d = hello/ test/
.: $d

# hello/buildfile
#
exe{hello}: cxx{hello utility}

# test/buildfile
#
exe{test}: cxx{test ../hello/utility}
```

Directory Scopes

Directory Scopes

```
/:
{
  home/boris/hello/:
  {
    hello/:
    {
      exe{hello}: cxx{hello} cxx{utility}
    }

    test/:
    {
      exe{test}: cxx{test} ../hello/cxx{utility}
    }

    dir{}: dir{hello/} dir{test/}
  }
}
```

Directory Scopes

```
/:
{
  home/boris/hello/:
  {
    hello/:
    {
      exe{hello}: cxx{hello} cxx{utility}
    }

    test/:
    {
      exe{test}: cxx{test} ../hello/cxx{utility}
    }

    dir{}: dir{hello/} dir{test/}
  }
}
```

Directory Scopes

```
/:
{
  home/boris/hello/:
  {
    hello/:
    {
      exe{hello}: cxx{hello} cxx{utility}
    }

    test/:
    {
      exe{test}: cxx{test} ../hello/cxx{utility}
    }

    dir{}: dir{hello/} dir{test/}
  }
}
```

Directory Scopes

```
/:
{
  home/boris/hello/:
  {
    hello/:
    {
      exe{hello}: cxx{hello} cxx{utility}
    }

    test/:
    {
      exe{test}: cxx{test} ../hello/cxx{utility}
    }

    dir{}: dir{hello/} dir{test/}
  }
}
```

Directory Scopes

```
/:
{
  home/boris/hello/:
  {
    hello/:
    {
      exe{hello}: cxx{hello} cxx{utility}
    }

    test/:
    {
      exe{test}: cxx{test} ../hello/cxx{utility}
    }

    dir{}: dir{hello/} dir{test/}
  }
}
```

Out-of-Tree Builds

```
hello/
└── ...

hello-gcc/
└── ...

hello-clang/
└── ...
```

Configuration

Configuration

```
using config
```

Configuration

```
using config


# bootstrap.build
#
project = hello
using config
```

Configuration Storage

```
save hello-clang/build/config.build
```

Configuration Storage

```
save hello-clang/build/config.build


# Created automatically by the config module, but
# feel free to edit.
#
config.cxx = clang++
config.cxx.poptions =
config.cxx.coptions =
config.cxx.loptions =
config.cxx.libs =
```

Configuration Specification

```
$ b config.cxx=clang++ config.cxx.coptions=-O3 configure
```

Configuration Specification

```
$ b config.cxx=clang++ config.cxx.coptions=-O3 configure

$ emacs build/config.build
```

Configuration Specification

```
$ b config.cxx=clang++ config.cxx.coptions=-O3 configure

$ emacs build/config.build

$ b config=”clang3.6 release” configure
```

Still to be Explained

- Operations
- Config module in bootstrap

Operations

```
operation(target: prerequisite1 prerequisite2 ...)
```

Operations

```
operation(target: prerequisite1 prerequisite2 ...)
```

```
update/clean
test
stage/unstage
install/uninstall
dist
```

Configure/Disfigure

Configure/Disfigure

```
meta-operation(operation(target: prerequisite1 ...))
```

Meta-Operations

Meta-Operations

```
configure/disfigure
perform
dryrun
help
```

Meta-Operations

```
configure/disfigure
perform
dryrun
help
```

```
$ b
$ b update
$ b perform(update)
```

Operations and Modules

- Built-in meta-operations: `perform`
- Built-in operations: `update clean`
- Module can add operations and meta-operations

Building Libraries

- Static, shared, or both
- Position-independent code (`-fPIC`)
- Linking priority

“Hello, World” Library

```
hello/
├── build/
│   ├── bootstrap.build
│   └── root.build
├── hello/
│   ├── buildfile
│   ├── hello.cxx
│   ├── utility.cxx
│   └── utility.hxx
│   test/
│   ├── buildfile
│   └── test.cxx
└── buildfile
```

Old hello/buildfile

```
# hello/buildfile
#
exe{hello}: cxx{hello utility}
```

New hello/buildfile

```
# hello/buildfile
#
lib{hello}: cxx{utility}

exe{hello}: cxx{hello} lib{hello}

.: exe{hello} lib{hello}
```

New hello/buildfile

```
# hello/buildfile
#
lib{hello}: cxx{utility}

exe{hello}: cxx{hello} lib{hello}

.: exe{hello} lib{hello}
```

New hello/buildfile

```
# hello/buildfile
#
lib{hello}: cxx{utility}

exe{hello}: cxx{hello} lib{hello}

.: exe{hello} lib{hello}
```

New hello/buildfile

```
# hello/buildfile
#
lib{hello}: cxx{utility}

exe{hello}: cxx{hello} lib{hello}

.: exe{hello} lib{hello}
```

Old test/buildfile

```
# test/buildfile
#
exe{test}: cxx{test ../hello/utility}
```

New test/buildfile

```
# test/buildfile
#
exe{test}: cxx{test} ../hello/lib{hello}
```

New test/buildfile

```
# test/buildfile
#
exe{test}: cxx{test} ../hello/lib{hello}

include ../hello/
```

Library/Object Target Types

Library/Object Target Types

- `obj{} lib{}` are target groups
- `obj{}: obja{} objso{}`
- `lib{}: liba{} libso{}`

Default Library Link Order

```
config.bin.exe.lib   = shared static
config.bin.liba.lib  = static
config.bin.libso.lib = shared
```

Group Member-Specific Customizations

```
lib{hello}: obj{utility}

obj{utility}: cxx{utility}
objso{utility}: cxx{utility-so}
```

Group Member-Specific Customizations

```
lib{hello}: obj{utility}

obj{utility}: cxx{utility}
objso{utility}: cxx{utility-so}
```

Group Member-Specific Customizations

```
lib{hello}: obj{utility}

obj{utility}: cxx{utility}
objso{utility}: cxx{utility-so}
```

Group Member-Specific Customizations

```
lib{hello}: obj{utility}

obj{utility}: cxx{utility}
objso{utility}: cxx{utility-so}
```

“Hello, World” Separate Library

```
libhello/
├── build/
│   ├── bootstrap.build
│   └── root.build
├── hello/
│   ├── buildfile
│   ├── utility.cxx
│   └── utility.hxx
├── test/
│   ├── buildfile
│   └── test.cxx
└── buildfile
```

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

libhello

```
# build/bootstrap.build
project = libhello
using config

# build/root.build
using cxx
cxx.poptions += -I$src_root

# hello/buildfile
lib{hello}: cxx{utility}

# test/buildfile
exe{test}: cxx{test} ../hello/lib{hello}
include ../hello/

# buildfile
d = hello/ test/
.: $d
include $d
```

hello

```
# build/bootstrap.build
project = hello
using config

# build/root.build
using cxx

# buildfile
exe{hello}: cxx{hello} lib{hello}
```

hello

```
# build/bootstrap.build
project = hello
using config

# build/root.build
using cxx

# buildfile
exe{hello}: cxx{hello} lib{hello}
```

libhello/test/buildfile

```
#
# test/buildfile
exe{test}: cxx{test} ../hello/lib{hello}
include ../hello/
```

libhello/test/buildfile

```
#
# test/buildfile
exe{test}: cxx{test} ../hello/lib{hello}
include ../hello/
```

libhello/test/buildfile

```
#
# test/buildfile
exe{test}: cxx{test} ../hello/lib{hello}
include ../hello/
```

Project Dependencies

Project Dependencies

- Put `-lhello` in `cxx.libs` and assume project is installed

Project Dependencies

- Put `-lhello` in `cxx.libs` and assume project is installed
- Put `-lhello` in `cxx.libs` and supply `-I/-L`

Project Dependencies

- Put `-lhello` in `cxx.libs` and assume project is installed
- Put `-lhello` in `cxx.libs` and supply `-I/-L`
- Bundle all prerequisite projects

Project Dependencies

- Put `-lhello` in `cxx.libs` and assume project is installed
- Put `-lhello` in `cxx.libs` and supply `-I/-L`
- Bundle all prerequisite projects
- Support project dependencies at build system level

hello/buildfile

```
import lh = libhello

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

hello/buildfile

```
import lh = libhello

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

hello/buildfile

```
import lh = libhello

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

Export Stub

```
libhello/
├── build/
│   ├── bootstrap.build
│   ├── export.build
│   └── root.build
│
...
```

Export Stub

```
libhello/
├── build/
│   ├── bootstrap.build
│   ├── export.build
│   └── root.build
│
...
```

Export Stub

```
# build/export.build
#
$out_root/:
{
  include hello/
}

export $out_root/hello/lib{hello}
```

Export Stub

```
# build/export.build
#
$out_root/:
{
  include hello/
}

export $out_root/hello/lib{hello}
```

Export Stub

```
# build/export.build
#
$out_root/:
{
  include hello/
}

export $out_root/hello/lib{hello}
```

Export Stub

```
# build/export.build
#
$out_root/:
{
  include hello/
}

export $out_root/hello/lib{hello}
```

Target-Specific “Export” Variables

```
cxx.export.poptions =
cxx.export.coptions =
cxx.export.loptions =
cxx.export.libs =
```

libhello/hello/buildfile

```
lib{hello}: cxx{utility}
lib{hello}: cxx.export.poptions = -I$src_root
```

libhello/hello/buildfile

```
lib{hello}: cxx{utility}
lib{hello}: cxx.export.poptions = -I$src_root
```

Implementation Details

Who Implements All of This?

Who Implements All of This?

Is it hard-coded into build2?

Modules

Modules

Modules can:

- Register new meta/operations (e.g., `configure`)

Modules

Modules can:

- Register new meta/operations (e.g., `configure`)
- Register new target types (e.g., `exe{}`)

Modules

Modules can:

- Register new meta/operations (e.g., `configure`)
- Register new target types (e.g., `exe{}`)
- Register new rules

Rule

Executes Meta/Operation on Target

Module Language

Module Language

What language is:

Module Language

What language is:

- Full-featured programming language

Module Language

What language is:

- Full-featured programming language
- Cross-platform

Module Language

What language is:

- Full-featured programming language
- Cross-platform
- Efficient

Module Language

What language is:

- Full-featured programming language
- Cross-platform
- Efficient
- Most C++ programmers already know

Module Language

Modules are Implemented in C++11

State of Implementation

State of Implementation

- Everything I have shown actually works
- Most of the internal “world view” established

State of Implementation

- Everything I have shown actually works
- Most of the internal “world view” established
- Not yet ready for production use, still long road

What's Next?

- Amalgamation and subprojects
- `test` `install` operations
- Automatic module building/loading
- Windows/VC++ support
- Inline C++ recipes
- Parallelism

Questions?

codesynthesis.com/projects/build2/