Boost-Python

First take

Getting Started on Windows

A note to Cygwin and MinGW users

If you plan to use your tools from the Windows command prompt, you're in the right place. If you plan to build from the Cygwin bash shell, you're actually running on a POSIX platform and should follow the instructions for getting started on Unix variants. Other command shells, such as MinGW's MSYS, are not supported—they may or may not work.

Index

Getting Started on Unix Variants

Index

1 Get Boost

The most reliable way to get a copy of Boost is to download a distribution from SourceForge:

1. Download `boost_1_57_0.tar.bz2`.
2. In the directory where you want to put the Boost installation, execute

```
tar --bzip2 -xf /path/to/boost_1_57_0.tar.bz2
```

2 The Boost Distribution

This is a sketch of the resulting directory structure:

```
boost_1_57_0/ ..................The "boost root directory"
   index.htm .........A copy of www.boost.org starts here
   boost/ ........................All Boost Header files

   libs/ ............Tests, .cpps, docs, etc., by library
     index.html ........Library documentation starts here
     algorithm/
     any/
     array/
                    …more libraries…
   status/ ........................Boost-wide test suite
   tools/ ...........Utilities, e.g. Boost.Build, quickbook, bcp
   more/ ..........................Policy documents, etc.
   doc/ ..............A subset of all Boost library docs
```

It's important to note the following:

1. The path to the **boost root directory** (often `/usr/local/boost_1_57_0`) is sometimes referred to as `$BOOST_ROOT` in documentation and mailing lists .

2. To compile anything in Boost, you need a directory containing the `boost/` subdirectory in your `#include` path.

3. Since all of Boost's header files have the `.hpp` extension, and live in the `boost/` subdirectory of the boost root, your Boost `#include` directives will look like:

   ```
   #include <boost/whatever.hpp>
   ```

   or

   ```
   #include "boost/whatever.hpp"
   ```

   depending on your preference regarding the use of angle bracket includes.

4. Don't be distracted by the `doc/` subdirectory; it only contains a subset of the Boost documentation. Start with `libs/index.html` if you're looking for the whole enchilada.

Header Organization

The organization of Boost library headers isn't entirely uniform, but most libraries follow a few patterns:

- Some older libraries and most very small libraries place all public headers directly into `boost/`.

- Most libraries' public headers live in a subdirectory of `boost/`, named after the library. For example, you'll find the Python library's `def.hpp` header in

  `boost/python/def.hpp`.

- Some libraries have an "aggregate header" in `boost/` that `#includes` all of the library's other headers. For example, Boost.Python's aggregate header is

  `boost/python.hpp`.

- Most libraries place private headers in a subdirectory called `detail/`, or `aux_/`. Don't expect to find anything you can use in these directories.

Just get the source, and compile Boost yourself it has become very easy. Here is an example for the current version of Boost (1.50.0) on the current OSX (10.7.4) as of this writing:

1. Download the the .tar.gz from http://sourceforge.net/projects/boost/files/boost/1.50.0/
2. Unpack and go into the directory:

```
tar -xzf boost_1_50_0.tar.gz
cd boost_1_50_0
```

3. Configure (and build `bjam`):

```
./bootstrap.sh --prefix=/some/dir/you/would/like/to/prefix
```

4. Build:

```
./b2
```

5. Install:

```
./b2 install
```

Depending on the prefix you choose in Step 3, you might need to sudo Step 5, if the script tries copy files to a protected location.

```
// Copyright Ralf W. Grosse-Kunstleve 2002-2004. Distributed under the Boost
// Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)

#include <boost/python/module.hpp>
#include <boost/python/def.hpp>
#include <string>

namespace { // Avoid cluttering the global namespace.

  // A couple of simple C++ functions that we want to expose to Python.
  std::string greet() { return "hello, world"; }
  int square(int number) { return number * number; }
}

namespace python = boost::python;

// Python requires an exported function called init<module-name> in every
// extension module. This is where we build the module contents.
BOOST_PYTHON_MODULE(getting_started1)
{
    // Add regular functions to the module.
    python::def("greet", greet);
    python::def("square", square);
}
```

on standard UNIX

```
# location of the Python header files

PYTHON_VERSION = 2.7
PYTHON_INCLUDE = /usr/include/python$(PYTHON_VERSION)

# location of the Boost Python include files and library

BOOST_INC = /usr/include
BOOST_LIB = /usr/lib

# compile
TARGET = hello_ext

$(TARGET).so: $(TARGET).o
   g++ -shared -Wl,--export-dynamic $(TARGET).o -L$(BOOST_LIB) -lboost_python-$
(PYTHON_VERSION) -L/usr/lib/python$(PYTHON_VERSION)/config -lpython$(PYTHON_VERSION) -o $
(TARGET).so

$(TARGET).o: $(TARGET).C
   g++ -I$(PYTHON_INCLUDE) -I$(BOOST_INC) -fPIC -c $(TARGET).C
```

on macs

```
   clang++ -shared -undefined dynamic_lookup $(TARGET).o -L$(BOOST_LIB) -lboost_python -L/
usr/lib/python$(PYTHON_VERSION)/config -lpython$(PYTHON_VERSION) -o $(TARGET).so

$(TARGET).o: $(TARGET).C
   clang++ -I$(PYTHON_INCLUDE) -I$(BOOST_INC) -fPIC -c $(TARGET).C
```

http://mrbook.org/blog/tutorials/make/