-----------------------------------------------------------------------

Creating a Cross-Platform Build System Using Bakefile

The 10-minute, do-it-yourself wx project baking guide (with free sample recipes!)

Status: DRAFT

Author: Kevin Ollivier

Date: 2/13/04

Licence: wxWindows Licence

-----------------------------------------------------------------------

Supporting many different platforms can be a difficult challenge. The

challenge for wxWidgets is especially great, because it supports a variety of

different compilers and development environments, including MSVC, Borland C++,

MinGW, DevCPP, GNU make/automake, among others. Maintaining such a large

number of different project files and formats can quickly become overwhelming.

To simplify the maintenance of these formats, one of the wxWidgets developers,

Vaclav Slavik, created Bakefile, a XML-based makefile wrapper that generates

all the native project files for wxWidgets. So now, even though wxWidgets

supports all these formats, wxWidgets developers need only update one file -

the Bakefile, and it handles the rest. But Bakefile isn't specific to

wxWidgets in any way - you can use Bakefile for your own projects, too. This

brief tutorial will take a look at how to do that.

Note that this tutorial assumes that you are familiar with how to build

software using one of the supported Bakefile makefile systems, that you have

some basic familiarity with how makefiles work, and that you are capable of

setting environment variables on your platform. Also note that the terms Unix

and Unix-based refers to all operating systems that share a Unix heritage,

including FreeBSD, Linux, Mac OS X, and various other operating systems.

-- Getting Started --

First, you'll need to install Bakefile. You can always find the latest version

for download online at http://www.bakefile.org. A binary installer is provided

for Windows users, while users of Unix-based operating systems (OS) will need

to unpack the tarball and run configure && make && make install. (binary

packages for some Linux distributions are also available, check

http://www.bakefile.org/download.html for details).

-- Setting Up Your wx Build Environment --

Before you can build wxWidgets software using Bakefile or any other build

system, you need to make sure that wxWidgets is built and that wxWidgets

projects can find the wxWidgets includes and library files. wxWidgets build

instructions can be found by going to the docs subfolder, then looking for the

subfolder that corresponds to your platform (i.e. msw, gtk, mac) and reading

"install.txt" there. Once you've done that, here are some extra steps you

should take to make sure your Bakefile projects work with wxWidgets:

On Windows

----------

Once you've built wxWidgets, you should create an environment variable named

WXWIN and set it to the home folder of your wxWidgets source tree. (If you use

the command line to build, you can also set or override WXWIN at build time by

passing it in as an option to your makefile.)

On Unix

-------

In a standard install, you need not do anything so long as wx-config is on

your PATH. wx-config is all you need. (See the section of the book on using

wx-config for more information.)

-- A Sample wx Project Bakefile --

Now that everything is setup, it's time to take Bakefile for a test run. I

recommend that you use the wx sample Bakefile to get you started. It can be

found in the 'build/bakefiles/wxpresets/sample' directory in the wxWidgets

source tree. Here is the minimal.bkl Bakefile used in the sample:

minimal.bkl

-------------------------------------------------------------

<?xml version="1.0" ?>

<makefile>

<include file="presets/wx.bkl"/>

<exe id="minimal" template="wxgui">

<debug-info>on</debug-info>

<runtime-libs>dynamic</runtime-libs>

<sources>minimal.cpp</sources>

<wx-lib>core</wx-lib>

<wx-lib>base</wx-lib>

</exe>

</makefile>

---------------------------------------------------------------

It's a complete sample ready to be baked, so go into the directory mentioned

above and run the following command:

On Windows:

bakefile -f msvc -I.. minimal.bkl

On Unix:

bakefile -f gnu -I.. minimal.bkl

It should generate a makefile (makefile.vc or GNUmakefile, respectively) which

you can use to build the software. Just build the software using the command

"nmake -f makefile.vc" or "make -f GNUmakefile" respectively. Now let's take a

look at some of the basic Bakefile concepts that you'll need to know to move

on from here.

-- Project Types --

As mentioned earlier, Bakefile builds makefiles for many different

development environments. The -f option accepts a list of formats that you

would like to build, separated by commas. Valid values are:

autoconf GNU autoconf Makefile.in files

borland Borland C/C++ makefiles

gnu GNU toolchain makefiles (Unix)

mingw MinGW makefiles (mingw32-make)

msvc MS Visual C++ nmake makefiles

TIP: autoconf Project Type

---------------------------

You may notice that in the sample folder, there is also a file called

configure.in. That file is the input for autoconf, which creates the configure

scripts that you often see when you build software from source on Unix-based

platforms. People use configure scripts because they make your Unix makefiles

more portable by automatically detecting the right libraries and commands to

use on the user's machine and OS. This is necessary because there are many

Unix-based operating systems and they all are slightly different in various

small ways.

Bakefile does not generate a configure or configure.in script, so if you want

to use configure scripts with your Unix-based software, you will need to learn

how to use autoconf. Unfortunately, this topic deserves a book all its own and

is beyond the scope of this tutorial, but a book on the subject can be found

online at: http://sources.redhat.com/autobook/. Note that you do not need to

use automake when you are using Bakefile, just autoconf, as Bakefile

essentially does the same thing as automake.

----------------------------

-- Targets --

Every project needs to have a target or targets, specifying what is to be

built. In Bakefile, you specify the target by creating a tag named with the

target type. The possible names for targets are:

exe create an executable file

dll create a shared library

lib create a static library

module create a library that is loaded at runtime (i.e. a plugin)

Note the sample above is an "exe" target. Once you create the target, all the

build settings, including flags and linker options, should be placed inside

the target tag, as they are in the sample above.

-- Adding Sources and Includes --

Obviously, you need to be able to add source and include files to your

project. You add sources using the "<sources>" tag (as shown above), and add

include directories using the "<include>" tag. You can add multiple <sources>

and <include> tags to add multiple source files, or you can also add multiple

sources and includes into one tag by separating them with a space, like so:

<sources>minimal.cpp minimal2.cpp minimal3.cpp</sources>

If your sources are in a subfolder of your Bakefile, you use the slash "/"

character to denote directories, even on Windows. (i.e. src/minimal.cpp) For

more options and flags, please consult the Bakefile documentation in the 'doc'

subfolder of Bakefile, or you can also find it on the Bakefile web site.

-- Build Options --

What if you want to offer a DEBUG and a RELEASE build? Or a UNICODE/ANSI

build? You can do this in Bakefile by creating options. To create an option,

use the "<option>" tag. A typical option has three important parts: a name, a

default value, and a comma-separated list of values. For example, here is how

to create a DEBUG option which builds debug by default:

<option name="DEBUG">

<default-value>1</default-value>

<values>0 1</values>

</option>

You can then test the value of this option and conditionally set build

settings, flags, etc. For more information on both options and conditional

statements, please refer to the Bakefile documentation.

-- Bakefile Presets/Templates and Includes --

It is common that most projects will reuse certain settings, or options, in

their makefiles. (i.e. DEBUG or static/dynamic library options) Also, it is

common to have to use settings from another project; for example, any project

that uses wxWidgets will need to build using the same flags and options that

wxWidgets was built with. Bakefile makes these things easier by allowing users

to create Bakefile templates, where you can store common settings.

Bakefile ships with a couple of templates, found in the 'presets' subfolder of

your Bakefile installation. The "simple.bkl" template adds a DEBUG option to

makefiles so you can build in release or debug mode. To add this template to

your project, simply add the tag "<include file="presets/simple.bkl"/>" to the

top of your Bakefile. Then, when creating your target, add the

"template="simple"" attribute to it. Now, once you build the makefile, your

users can write commands like:

nmake -f makefile.vc DEBUG=1

or

make -f GNUmakefile DEBUG=1

In order to build the software in debug mode.

To simplify the building of wxWidgets-based projects, wxWidgets contains a

set of Bakefiles that automatically configure your build system to be

compatible with wxWidgets. As you'll notice in the sample above, the sample

project uses the "wxgui" template. Once you've included the template, your software

will now build as a GUI application with wxWidgets support.

There's also "wxconsole" template for building console-based wxWidgets applications

and "wx" template that doesn't specify application type (GUI or console) and can be

used e.g. for building libraries that use wxWidgets.

But since the wx presets don't exist in the Bakefile presets subfolder,

Bakefile needs to know where to find these presets. The "-I" command adds the

wxpresets folder to Bakefile's search path.

If you regularly include Bakefile presets in places other than the Bakefile

presets folder, then you can set the BAKEFILE\_PATHS environment variable so

that Bakefile can find these Bakefiles and include them in your project. This

way you no longer need to specify the -I flag each time you build.

Lastly, it's important to note that the Win 32 wx project Bakefiles come with

some common build options that users can use when building the software. These

options are:

Option Values Description

------ ------ -------------

WX\_MONOLITHIC 0(default),1 Set this to 1 if you built wx

as a monolithic library

WX\_SHARED 0(default),1 Specify static or dynamic wx libs

WX\_UNICODE 0(default),1 Use ANSI or UNICODE wx libs

WX\_DEBUG 0,1(default) Use release or debug wx libs

\*WX\_VERSION 25,26(default) Specify version of wx libs

\*Note: Any version of wx past 2.5 will be allowed here, so 25/26 is not a

complete list of values.

These options are not needed under Unix as wx-config can be used to specify

these options.

-- bakefile\_gen - Automated Bakefile Scripts --

If you have a large project, you can imagine that the calls to Bakefile would

get more and more complex and unwieldy to manage. For this reason, a script

called bakefile\_gen was created, which reads in a .bkgen file that provides

all the commands needed to build all the makefiles your project supports. A

discussion of how to use bakefile\_gen is beyond the scope of this tutorial,

but it deserves mention because it can be invaluable to large projects.

Documentation on bakefile\_gen can be found in the Bakefile documentation.

-- Conclusion --

This concludes our basic tutorial of the cross-platform Bakefile build system

management tool. From here, please be sure to take a good look at the Bakefile

documentation to see what else it is capable of. Please post questions to the

bakefile-devel@lists.sourceforge.net list, or if you have questions specific

to the wx template Bakefile, send an email to the wxWidgets users mailing list:

https://www.wxwidgets.org/support/mailing-lists/

Enjoy using Bakefile!