wxWidgets for OS X installation

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

wxWidgets can be compiled using Apple's Cocoa library.

Most OS X developers should start by downloading and installing Xcode

from the App Store. It is a free IDE from Apple that provides

all of the tools you need for working with wxWidgets.

After Xcode is installed, download wxWidgets-{version}.tar.bz2 and then

double-click on it to unpack it to create a wxWidgets directory.

Next use Terminal (under Applications, Utilities, Terminal) to access a command

prompt. Use cd to change directories to your wxWidgets directory and execute

the following sets of commands from the wxWidgets directory.

---------

mkdir build-cocoa-debug

cd build-cocoa-debug

../configure --enable-debug

make

# Build the samples and demos

cd samples; make;cd ..

cd demos; make;cd ..

---------

After the compilation completes, use Finder to run the samples and demos

Go to build-cocoa-debug/samples to experiment with the Cocoa samples.

Go to build-cocoa-debug/demos to experiment with the Cocoa demos.

Double-click on the executables which have an icon showing three small squares.

The source code for the samples is in wxWidgets/samples

The source code for the demos is in wxWidgets/demos

---------

More information about building on OS X is available in the wxWiki.

Here are two useful links

https://wiki.wxwidgets.org/Guides\_%26\_Tutorials

https://wiki.wxwidgets.org/Development:\_wxMac

---------

More advanced topics are covered below.

---------

If you want to install the library into the system directories you'll need

to do this as root. The accepted way of running commands as root is to

use the built-in sudo mechanism. First of all, you must be using an

account marked as a "Computer Administrator". Then

6) sudo make install

7) type <YOUR OWN PASSWORD>

Note that while using this method is okay for development, it is not

recommended that you require endusers to install wxWidgets into their

system directories in order to use your program. One way to avoid this

is to configure wxWidgets with --disable-shared. Another way to avoid

it is to make a framework for wxWidgets. Making frameworks is beyond

the scope of this document.

Note:

It is rarely desirable to install non-Apple software into system directories.

By configuring the library with --disable-shared and using the full path

to wx-config with the --in-place option you can avoid installing the library.

Apple Developer Tools: Xcode

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

You can use the project in build/osx/wxcocoa.xcodeproj to build the Cocoa

version of wxWidgets (wxOSX/Cocoa). There are also sample

projects supplied with the minimal sample.

Notice that the command line build above builds not just the library itself but

also wxrc tool which doesn't have its own Xcode project. If you need this tool,

the simplest possibility is to build it from the command line after installing

the libraries using commands like this:

$ cd utils/wxrc

$ g++ -o wxrc wxrc.cpp `wx-config --cxxflags --libs base,xml`

Creating universal binaries

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

The Xcode projects for the wxWidgets library and minimal project are set up

to create universal binaries.

If using the Apple command line tools, pass --enable-universal\_binary when

configuring wxWidgets. This will create the libraries for all the supported

architectures, currently ppc, i386 and x86\_64 . You may explicitly specify

the architectures to use as a comma-separated list,

e.g. --enable-universal\_binary=i386,x86\_64.

Notice that if you use wx-config --libs to link your application, the -arch

flags are not added automatically as it is possible to link e.g. x86\_64-only

program to a "fat" library containing other architectures. If you want to

build a universal application, you need to add the necessary "-arch xxx" flags

to your project or makefile separately.

As an alternative to using --enable-universal\_binary, you can build for

each architecture separately and then use the lipo tool to glue the

binaries together. Assuming building on a PPC system:

1. First build in the usual way to get the PPC library.

2. Then, build for Intel, in a different folder. This time use:

export CFLAGS="-g -isysroot /Developer/SDKs/MacOSX10.7.sdk -arch i386"

export LDFLAGS="-syslibroot,/Developer/SDKs/MacOSX10.7.sdk"

./configure --disable-dependency-tracking --enable-static=yes --enable-shared=no \

--target=i386-apple-darwin8 --host=powerpc-apple-darwin8 --build=i386-apple-darwin8

You will need to reverse the powerpc and i386 parameters everywhere to build PPC on an Intel

machine.

3. Use lipo to glue the binaries together.

See also:

http://developer.apple.com/technotes/tn2005/tn2137.html