Building wxPython 2.5 for Development and Testing

This file describes how I build wxWidgets and wxPython while doing development and testing, and is meant to help other people that want to do the same thing. I'll assume that you are using either a CVS snapshot from http://wxWidgets.org/snapshots/, a checkout from CVS, or one of the released wxPython-src-2.5.* tarballs. I'll also assume that you know your way around your system, the compiler, etc. and most importantly, that you know what you are doing! ;-)

If you want to also install the version of wxPython you build to be in your site-packages dir and be your default version of wxPython, then a few additional steps are needed, and you may want to use slightly different options. See the INSTALL document for more details. If you only use the instructions in this BUILD document file then you will end up with a separate installation of wxPython and you can switch back and forth between this and the release version that you may already have installed.

If you want to make changes to any of the *.i files (SWIG interface definition files,) or to regenerate the extension sources or renamer modules, then you will need an up to date version of SWIG, plus some patches. Get the sources for version 1.3.22, and then apply the patches in wxPython/SWIG and then build SWIG like normal. See the README.txt in the wxPython/SWIG dir for details about each patch and also info about those that may already have been applied to the SWIG sources. If you install this build of SWIG to a location that is not on the PATH (so it doesn't interfere with an existing SWIG install for example) then you can set a setup.py command-line variable named SWIG to be the full path name of the executable and the wxPython build will use it. See below for an example.

In the text below I'll use WXDIR with environment variable syntax (either $WXDIR or %WXDIR%) to refer to the top level directory where your wxWidgets and wxPython sources are located. It will equate to wherever you checked out the wxWidgets module from CVS, or untarred the wxPython-src tarball to. You can either substitute the $WXDIR text below with your actual dir, or set the value in the environment and use it just like you see it below.

If you run into what appears to be compatibility issues between wxWidgets and wxPython while building wxPython, be sure you are using the wxWidgets sources included with the wxPython-src tarball or the CVS snapshot, and not a previously installed version or a version installed from one of the standard wxWidgets installers. With the "unstable" releases (have a odd-numbered minor release value, where the APIs are allowed to change) there are often significant differences between the W.X.Y release of wxWidgets and the W.X.Y.Z release of wxPython.

Building on Unix-like Systems (e.g. Linux and OS X)

These platforms are built almost the same way while in development so I'll combine the descriptions about their build process here. First we will build wxWidgets and install it to an out of the way place, then do the same for wxPython.

1. Create a build directory in the main wxWidgets dir, and configure wxWidgets. If you want to have multiple builds with different configure options, just use different subdirectories. I normally put the configure command in a script named ".configure" in each build dir so I can easily blow away everything in the build dir and rerun the script without having to remember the options I used before:

   ```
   cd $WXDIR
   mkdir bld
   cd bld
   ../configure --prefix=/opt/wx/2.5 \
                --with-gtk \
                --with-opengl \
                --enable-debug \
                --enable-geometry \
                --enable-sound --with-sdl \
                --enable-display \
   ```

   On OS X of course you'll want to use --with-mac instead of --with-gtk.

   NOTE: Due to a recent change there is currently a dependency problem in the multilib builds of wxWidgets on OSX, so I have switched to using a monolithic build. That means that all of the core wxWidgets code is placed in in one shared library instead of several. wxPython can be used with either mode, so use whatever suits you on Linux and etc. but use monolithic on OSX. To switch to the monolithic build of wxWidgets just add this configure flag:

   ```
   --enable-monolithic \
   ```

   By default GTK2 will be selected if its development package is installed on your build system. To force the use of GTK 1.2.x instead add this flag:

   ```
   --disable-gtk2 \
   ```

   To make the wxWidgets build be unicode enabled (strongly recommended if you are building with GTK2) then add the following. When wxPython is unicode enabled then all strings that are passed to wx functions and methods will first be converted to unicode objects, and any 'strings' returned from wx functions and methods will actually be unicode objects.:

   ```
   --enable-unicode \
   ```

   Notice that I used a prefix of /opt/wx/2.5. You can use whatever path you want, such as a path in your HOME dir or even one of the standard prefix paths such as /usr or /usr/local if you like, but using /opt this way lets me easily have multiple versions and ports of wxWidgets "installed" and makes it easy to switch between them, without impacting any versions of wxWidgets that may have been installed via an RPM or whatever. For the rest of the steps below be sure to also substitute "/opt/wx/2.5" with whatever prefix you choose for your build.

   If you want to use the image and zlib libraries included with wxWidgets instead of those already installed on your system, (for example, to reduce dependencies on 3rd party libraries) then you can add these flags to the configure command:

   ```
   --with-libjpeg=builtin \
   --with-libpng=builtin \
   --with-libtiff=builtin \
   --with-zlib=builtin \
   ```

2. To build and install wxWidgets you could just use the "make" command but there are other libraries besides the main wxWidgets libs that also need to be built so again I make a script to do it all for me so I don't forget anything. This time it is called ".make" (I use the leading "." so when I do `rm -r *` in my build dir I don't lose my scripts too.) This is what it looks like:

   ```
   make $* \
       && make -C contrib/src/gizmos $* \
       && make -C contrib/src/ogl CXXFLAGS="-DwxUSE_DEPRECATED=0" $* \
       && make -C contrib/src/stc $*
   ```

   So you just use .make as if it where make, but don't forget to set the execute bit on .make first!:

   ```
   .make
   .make install
   ```

   When it's done you should have an installed set of files under /opt/wx/2.5 containing just wxWidgets. Now to use this version of wxWidgets you just need to add /opt/wx/2.5/bin to the PATH and set LD_LIBRARY_PATH (or DYLD_LIBRARY_PATH on OS X) to /opt/wx/2.5/lib.

3. I also have a script to help me build wxPython and it is checked in to the CVS as wxWidgets/wxPython/b, but you probably don't want to use it as it's very cryptic and expects that you want to run SWIG, so if you don't have the latest patched up version of SWIG then you'll probably get stuck. So I'll just give the raw commands instead.

   We're not going to install the development version of wxPython with these commands, so it won't impact your already installed version of the latest release. You'll be able test with this version when you want to, and use the installed release version the rest of the time. If you want to install the development version please read INSTALL.txt.

   If you have more than one version of Python on your system then be sure to use the version of Python that you want to use when running wxPython programs to run the setup.py commands below. I'll be using python2.3.

   Make sure that the first wx-config found on the PATH is the one you installed above, and then change to the $WXDIR/wxPython dir and run the this command:

   ```
   cd $WXDIR/wxPython
   python2.3 setup.py build_ext --inplace --debug
   ```

   If your new wx-config script is not on the PATH, or there is some other version of it found first, then you can add this to the command line to ensure your new one is used instead:

   ```
   WX_CONFIG=/opt/wx/2.5/bin/wx-config
   ```

   By default setup.py will assume that you built wxWidgets to use GTK2. If you built wxWidgets to use GTK 1.2.x then you should add this flag to the command-line:

   ```
   WXPORT=gtk
   ```

   If you would like to do a Unicode enabled build (all strings sent to or returned from wx functions are Unicode objects) and your wxWidgets was built with unicode enabled then add this flag:

   ```
   UNICODE=1
   ```

   If you are wanting to have the source files regenerated with swig, then you need to turn on the USE_SWIG flag and optionally tell it where to find the new swig executable, so add these flags:

   ```
   USE_SWIG=1 SWIG=/opt/swig/bin/swig
   ```

   If you get errors about being unable to find libGLU, wxGLCanvas being undeclared, or something similar then you can add BUILD_GLCANVAS=0 to the setup.py command line to disable the building of the glcanvas module.

   When the setup.py command is done you should have fully populated wxPython and wx packages locally in $WXDIR/wxPython/wxPython and $WXDIR/wxPython/wx, with all the extension modules (*.so files) located in the wx package.

4. To run code with the development version of wxPython, just set the PYTHONPATH to the wxPython dir in the source tree. For example:

   ```
   export LD_LIBRARY_PATH=/opt/wx/2.5/lib
   export PYTHONPATH=$WXDIR/wxPython
   cd $WXDIR/wxPython/demo
   python2.3 demo.py
   ```

   OS X NOTE: You need to use "pythonw" on the command line to run wxPython applications. This version of the Python executable is part of the Python Framework and is allowed to interact with the display. You can also double click on a .py or .pyw file from the finder (assuming that the PythonLauncher app is associated with these file extensions) and it will launch the Framework version of Python for you. For information about creating Applicaiton Bundles of your wxPython apps please see the wiki and the mail lists.

   SOLARIS NOTE: If you get unresolved symbol errors when importing wxPython and you are running on Solaris and building with gcc, then you may be able to work around the problem by uncommenting a bit of code in setup.py and building again. Look for 'SunOS' in setup.py and uncomment the block containing it. The problem is that Sun's ld does not automatically add libgcc to the link step.

Building on Windows

The Windows builds currently require the use of Microsoft Visual C++. Theoretically, other compilers (such as mingw32 or the Borland compilers) can also be used but I've never done the work to make that happen. If you want to try that then first you'll want to find out if there are any tricks that have to be done to make Python extension modules using that compiler, and then make a few changes to setup.py to accommodate that. (And send the patches to me.) If you plan on using VisualStudio.Net (a.k.a. MSVC 7.1) keep in mind that you'll also have to build Python and any other extension modules that you use with that compiler because a different version of the C runtime library is used. The Python executable that comes from PythonLabs and the wxPython extensions that I distribute are built with MSVC 6 with all the Service Packs applied. This policy will change with Python 2.4 and MSVC 7.1 will be used starting with that version.

If you want to build a debuggable version of wxWidgets and wxPython you will need to have also built a debug version of Python and any other extension modules you need to use. You can tell if you have them already if there is a _d in the file names, for example python_d.exe or python23_d.dll. If you don't need to trace through the C/C++ parts of the code with the debugger then building the normal (or hybrid) version is fine, and you can use the regular python executables with it.

Starting with 2.5.3.0 wxPython can be built for either the monlithic or the multi-lib wxWidgets builds. (Monolithic means that all the core wxWidgets code is in one DLL, and multi-lib means that the core code is divided into multiple DLLs.) To select which one to use specify the MONOLITHIC flag for both the wxWidgets build and the wxPython build as shown below, setting it to either 0 or 1.

Just like the unix versions I also use some scripts to help me build wxWidgets, but I use some non-standard stuff to do it. So if you have bash (cygwin or probably MSYS too) or 4NT plus unix-like cat and sed programs then there is a copy of my wxWidgets build scripts in %WXDIR%\wx\Python\distrib\msw. Just copy them to %WXDIR%\build\msw and you can use them to do your build, otherwise you can do everything by hand as described below. But if you do work by hand and something doesn't seem to be working correctly please refer to the build scripts to see what may need to be done differently.

The *.btm files are for 4NT and the others are for bash. They are:

```
.make/.make.btm       Builds the main lib and the needed contribs
.mymake/.mymake.btm   Builds just one lib, use by .make
.makesetup.mk         A makefile that will copy and edit setup.h
                      as needed for the different types of builds
```

Okay. Here's what you've been waiting for, the instructions! Adapt accordingly if you are using the bash shell.

1. Set an environment variable to the root of the wxWidgets source tree. This is used by the makefiles:

   ```
   set WXWIN=%WXDIR%
   ```

2. Copy setup0.h to setup.h:

   ```
   cd %WXDIR%\include\wx\msw
   copy setup0.h setup.h
   ```

3. Edit %WXDIR%\include\wx\msw\setup.h and change a few settings:

   ```
   wxDIALOG_UNIT_COMPATIBILITY    0
   wxUSE_DEBUG_CONTEXT            1
   wxUSE_MEMORY_TRACING           1
   wxUSE_DIALUP_MANAGER           0
   wxUSE_GLCANVAS                 1
   wxUSE_POSTSCRIPT               1
   wxUSE_AFM_FOR_POSTSCRIPT       0
   wxUSE_DISPLAY                  1
   ```

   If you are using my build scripts then a few more settings will be changed and then a copy of setup.h is placed in a subdir of %WXWIN%\lib\vc_dll. If you are doing it by hand and making a UNICODE build, then also change these:

   ```
   wxUSE_UNICODE                  1
   wxUSE_UNICODE_MSLU             1
   ```

   If you are doing a "hybrid" build (which is the same as the binaries that I release) then also change these:

   ```
   wxUSE_MEMORY_TRACING           0
   wxUSE_DEBUG_CONTEXT            0
   ```

4. Make sure that %WXDIR%\lib\vc_dll directory is on the PATH. The wxWidgets DLLs will end up there as part of the build and so you'll need it on the PATH for them to be found at runtime.

5. Change to the %WXDIR%\build\msw directory

   ```
   cd %WXDIR%\build\msw
   ```

6. If using my scripts then use the .make.btm command to build wxWidgets. It needs one command-line parameter which controls what kind of build(s) to do. Use one of the following:

   ```
   debug         Build debug version
   hybrid        Build hybrid version
   both          Both debug and hybrid
   debug-uni     Build a debug unicode library
   hybrid-uni    Hybrid unicode (see the pattern yet? ;-)
   both-uni      and finally both unicode libraries
   ```

   For example:

   ```
   .make hybrid
   ```

   You can also pass additional command line parameters as needed and they will all be passed on to the nmake commands, for example to clean up the build:

   ```
   .make hybrid clean
   ```

   If *not* using my scripts then you can do it by hand by directly executing nmake with a bunch of extra command line parameters. The base set are:

   ```
   -f makefile.vc OFFICIAL_BUILD=1 SHARED=1 MONOLITHIC=1 USE_OPENGL=1
   ```

   If doing a debug build then add:

   ```
   BUILD=debug
   ```

   otherwise add these:

   ```
   DEBUG_FLAG=1 CXXFLAGS="/D__NO_VC_CRTDBG__ WXDEBUGFLAG=h BUILD=release
   ```

   If doing a Unicode build then add these flags:

   ```
   UNICODE=1 MSLU=1
   ```

   Now, from the %WXDIR%\build\msw directory run nmake with your selection of command-line flags as described above. Repeat this same command from the following directories in order to build the contrib libraries:

   ```
   %WXDIR%\contrib\build\gizmos
   %WXDIR%\contrib\build\stc
   %WXDIR%\contrib\build\ogl
   ```

   ```
   Note, that the ogl lib build will need an additional flag::

       CPPFLAGS="-DwxUSE_DEPRECATED=0"
   ```

7. When that is all done it will have built the main wxWidgets DLLs and also some of the contribs DLLs. There should be a ton of DLLs and lots of lib files and other stuff in %WXDIR%\lib\vc_dll.

8. Building wxPython on Windows is very similar to doing it for the unix systems. We're not going to install the development version of wxPython with these commands, so it won't impact your already installed version of the latest release. You'll be able to test with this version when you want to, and use the installed release version the rest of the time. If you ever do want to install the development version please refer to INSTALL.txt.

   Change to the %WXDIR%\wxPython dir and run the this command, making sure that you use the version of python that you want to build for (if you have more than one on your system) and to match the MONOLITHIC flag with how you built wxWidgets:

   ```
   cd %WXDIR%\wxPython
   python setup.py build_ext --inplace MONOLITHIC=1
   ```

   If you are wanting to have the source files regenerated with swig, then you need to turn on the USE_SWIG flag and optionally tell it where to find the new swig executable, so add these flags:

   ```
   USE_SWIG=1 SWIG=e:\projects\SWIG-cvs\swig.exe
   ```

   If you built a Unicode version of wxWidgets and want to also build the Unicode version of wxPython then add this flag:

   ```
   UNICODE=1
   ```

   If you have a debug version of Python and wxWidgets and want to build a debug version of wxPython too, add the –debug flag to the command line. You should then end up with a set of *_d.pyd files in the wx package and you'll have to run python_d.exe to use them. The debug and hybrid(release) versions can coexist.

   When the setup.py command is done you should have fully populated wxPython and wx packages locally in %WXDIR%/wxPython/wxPython and %WXDIR%/wxPython/wx, with all the extension modules (*.pyd files) located in the wx package.

9. To run code with the development version of wxPython, just set the PYTHONPATH to the wxPython dir in the CVS tree. For example:

   ```
   set PYTHONPATH=%WXDIR%\wxPython
   cd %WXDIR%\wxPython\demo
   python demo.py
   ```