Installing wxWidgets for Windows

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

This is wxWidgets for Microsoft Windows (XP, Vista, 7, 8, 10, etc)

including both 32 bit and 64 bit versions.

Table of Contents:

- Installation

- Building wxWidgets

- Configuring the Build

- Building Applications Using wxWidgets

Installation

============

If you are using one of the supported compilers, you can download the

pre-built in binaries from

https://github.com/wxWidgets/wxWidgets/releases/v3.1.1

In this case, just uncompress the binaries archive under any directory

and skip to "Building Applications Using wxWidgets" part.

Otherwise, or if you want to build a configuration of the library

different from the default one, you need to build the library from

sources before using it.

The first step, which you may have already performed, unless you are

reading this file online, is to download the source archive and

uncompress it in any directory. It is strongly advised to avoid using

spaces in the name of this directory, i.e. notably do \*not\* choose a

location under "C:\Program Files", as this risks creating problems

with makefiles and other command-line tools.

After choosing the directory location, please define WXWIN environment

variable containing the full path to this directory. While this is not

actually required, this makes using the library more convenient and

this environment variable is used in the examples below.

NB: If you checked your sources from version control repository and

didn't obtain them from a release file, you also need to copy

include/wx/msw/setup0.h to include/wx/msw/setup.h and to remember

to update the latter whenever the former changes, otherwise you

will get compilation errors if any new symbols are added to

setup0.h file in the repository.

If you have no intention of modifying setup.h, you may avoid this

problem by creating a symbolic link to setup0.h instead of making

a copy of it using mklink, from an admin command prompt:

cd %WXWIN%\include\wx\msw\

mklink setup.h setup0.h

Building wxWidgets

==================

The following sections explain how to compile wxWidgets with each supported

compiler, see the "Building Applications" section about the instructions for

building your application using wxWidgets.

All makefiles and project are located in build\msw directory.

Microsoft Visual C++ Compilation

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

\* From command line using the provided makefiles:

0. Open a "Visual Studio Command Prompt" window shortcut to which

must have been installed to the "Start" menu or the "Start" screen

by MSVS installation.

1. Change directory to %WXWIN%\build\msw and type

> nmake /f makefile.vc

to build wxWidgets in the default debug configuration as a static

library. You can also do

> nmake /f makefile.vc BUILD=release

to build a release version or

> nmake /f makefile.vc BUILD=release SHARED=1

to build a release DLL version. Finally, you can also add

"TARGET\_CPU=X64" to nmake command line to build Win64 versions

(this only works if you are using a 64 bit compiler, of course).

See "Configuring the Build" for more information about the

additional parameters that can be specified on the command line.

2. To verify your build, change the directory to %WXWIN%\samples\minimal and

run the same nmake command (with the same parameters there), this

should create a working minimal wxWidgets sample.

3. If you need to rebuild, use "clean" target first or "nmake /a".

\* From the IDE using the provided project files:

Ready to use project files are provided for VC++ versions 7, 8, 9,

10, 11, 12, 14 and 15 (also known as MSVS 2003, 2005, 2008, 2010, 2012, 2013,

2015 and 2017 respectively).

Simply open wx\_vcN.sln (for N=7, 8, 9, 10, 11, 12, 14, or 15) file,

select the appropriate configuration (Debug or Release, static or DLL)

and build the solution. Notice that when building a DLL configuration,

you may need to perform the build several times because the projects

are not always built in the correct order, and this may result in link

errors. Simply do the build again, up to 3 times, to fix this.

Special notes for Visual Studio 2010+:

For Visual Studio 2010+ solutions it is possible to customize the build by

creating a wx\_local.props file in the build\msw directory which is used, if it

exists, by the projects. The settings in that file override the default values

for the properties such as wxCfg (corresponding to the CFG makefile variable

described below) or wxVendor (corresponding to VENDOR). The typical way to

make the file is to copy wx\_setup.props to wx\_local.props and then edit local.

For example, if you are building wxWidgets libraries using multiple versions

of Visual Studio you could change wxCompilerPrefix to include the toolset:

- <wxCompilerPrefix>vc</wxCompilerPrefix>

+ <wxCompilerPrefix>vc$(PlatformToolsetVersion)</wxCompilerPrefix>

Following that example if you are using Visual Studio 2013 and open

wx\_vc12.sln it will build using the "vc120" prefix for the build directories

so to allow its build files to coexist with the files produced by the other

MSVC versions.

Keep in mind that by using a separate local props file you ensure that your

changes won't be lost when updating to a future wxWidgets version. But if

wx\_setup.props is updated in some later commit your wx\_local.props is not

updated with it. For example the version information in wx\_setup.props could

change and the information in your wx\_local.props would be outdated. It is

your responsibility to monitor for such situations.

Improve debugging for Visual Studio 2012+:

Debug visualizers for Visual Studio 2012+ are provided which makes inspecting

various wxWidgets classes easier to view while debugging. To use them:

1. Open the folder %WXWIN%\misc\msvc

2. Open the folder %USERPROFILE%\My Documents\Visual Studio 2012\Visualizers\

(or the corresponding location for newer versions, e.g. ...2013\Visualizers)

3. Copy wxWidgets.natvis and autoexp.inc

4. For Visual Studio 2013+ additionally copy wxWidgets.2013.natvis

Cygwin/MinGW Compilation

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

wxWidgets supports Cygwin, MinGW, MinGW-w64 and TDM-GCC tool chains under

Windows. They can be downloaded from:

http://www.cygwin.com/

http://www.mingw.org/

http://mingw-w64.sourceforge.net/

http://tdm-gcc.tdragon.net/

respectively. Please retrieve and install the latest version of your preferred

tool chain by following the instructions provided by these packages. Notice

that Cygwin includes both native Cygwin compiler, which produces binaries that

require Cygwin during run-time, and MinGW[-w64] cross-compilers which can still

be used in Cygwin environment themselves but produce plain Windows binaries

without any special run-time requirements. You will probably want to use the

latter for developing your applications.

If using MinGW, you can download the add-on MSYS package to provide Unix-like

tools that you'll need to build wxWidgets using configure.

C++11 note: If you want to compile wxWidgets in C++11 mode, you currently have

to use -std=gnu++11 switch as -std=c++11 disables some extensions

that wxWidgets relies on. I.e. please use CXXFLAGS="-std=gnu++11".

All of these tool chains can be used either with Unix-like configure+make build

process (preferred) or with the provided makefile.gcc makefiles without using

configure:

\* Using configure

This method works in exactly the same way as under Unix systems,

including OS X, and requires a Unix-like environment to work, i.e.

either MSYS or Cygwin.

0. Open MSYS or Cygwin shell prompt.

1. Create a build directory: it is is strongly recommended to not

build the library in the directory containing the sources ($WXWIN)

but to create a separate build directory instead. The build

directory can be placed anywhere (using the fastest available disk

may be a good idea), but in this example we create it as a

subdirectory of the source one:

$ cd $WXWIN

$ mkdir build-debug

$ cd build-debug

2. Run configure passing it any of the options shown by "configure

--help". Notice that configure builds shared libraries by default,

use --disable-shared to build static ones. For example:

$ ../configure --enable-debug

3. Build the library:

$ make

4. Test the library build by building the minimal sample:

$ cd samples/minimal

$ make

5. Optionally install the library in a global location

$ make install

Notice that there is not much benefice to installing under Windows

so this step can usually be omitted.

\* Using plain makefiles:

NOTE: The makefile.gcc makefiles are for compilation under MinGW using

Windows command interpreter (command.com/cmd.exe), they won't work

if you use Unix shell, as is the case with MSYS. Follow the instructions

for using configure above instead if you prefer to use Unix shell.

0. Open DOS command line window (cmd.exe, \*not\* Bash sh.exe).

1. Change directory to %WXWIN%\build\msw and type

> mingw32-make -f makefile.gcc

to build wxWidgets in the default debug configuration as a static

library. Add "BUILD=release" and/or "SHARED=1" to build the library

in release configuration and/or as a shared library instead of the

default static one.

NOTE: For parallel builds, i.e. using -jN make option, please run

"mingw32-make -jN ... setup\_h" first and then rerun the full

make command without "setup\_h" at the end to work around a bug

in the current makefiles.

2. To verify your build, change the directory to samples\minimal and

run the same mingw32-make command (with the same parameters there),

this should create a working minimal wxWidgets sample.

3. If you need to rebuild, use "clean" target first.

Borland C++ Compilation

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

WARNING: Borland instructions are out of date, please send us your

corrections if you are using it with wxWidgets 3.0.

The minimum version required is 5.5 (last version supported by BC++ 5.0 was

2.4.2), which can be downloaded for free from:

http://www.borland.com/products/downloads/download\_cbuilder.html

We have found that the free Turbo Explorer and commercial BDS work fine; the

debugger is very good. To avoid linker errors you will need to add

-DSHARED=1 to the makefile line for the library

The version 5.6 included in Borland C++ Builder 2006 works as well after the

following small change: please remove the test for \_\_WINDOWS\_\_ from line 88

of the file BCCDIR\include\stl\\_threads.h.

Compiling using the makefiles:

1. Change directory to build\msw. Type 'make -f makefile.bcc' to

make the wxWidgets core library. Ignore the compiler warnings.

This produces a couple of libraries in the lib\bcc\_lib directory.

2. Change directory to a sample or demo such as samples\minimal, and type

'make -f makefile.bcc'. This produces a windows exe file - by default

in the bcc\_mswd subdirectory.

Note (1): the wxWidgets makefiles assume dword structure alignment. Please

make sure that your own project or makefile settings use the

same alignment, or you could experience mysterious crashes. To

change the alignment, change CPPFLAGS in build\msw\config.bcc.

Note (2): If you wish debug messages to be sent to the console in

debug mode, edit makefile.bcc and change /aa to /Tpe in link commands.

Using the Debugger and IDE in BDS or Turbo Explorer

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

Doubleclick / open %WXWIN%\samples\minimal\borland.bdsproj. The current version

is to be used with a dynamic build of wxWidgets-made by running

make -f Makefile.bcc -DBUILD=debug -DSHARED=1

in wxWidgets\build\msw. You also need the wxWidgets\lib\bcc\_dll

directory in your PATH. The debugger tracks your source and also

traces into the wxWidgets sources.

To use this to debug other samples, copy the borland\_ide.cpp

and borland.bdsproj files, then replace all occurrences of

"minimal" with the name of the new project files

Compilation succeeds with CBuilderX personal edition and CBuilder6, but

you may have to copy make.exe from the 5.5 download to the new bin directory.

Compiling using the IDE files for Borland C++ 5.0 and using CBuilder IDE

(v1-v6): not supported

\*\* REMEMBER \*\*

In all of your wxWidgets applications, your source code should include

the following preprocessor directive:

#ifdef \_\_BORLANDC\_\_

#pragma hdrstop

#endif

(check the samples -- e.g., \wx2\samples\minimal\minimal.cpp -- for

more details)

Configuring the Build

================================================================

NOTE: If you use configure to build the library with Cygwin/MinGW, the

contents of this section does not apply, just pass the arguments

to configure directly in this case.

Library configuration

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

While it is never necessary to do it, you may want to change some of

the options in the %WXWIN%\include\wx\msw\setup.h file before building

wxWidgets. This file is heavily commented, please read it and enable or disable

the features you would like to compile wxWidgets with[out].

Notice that this file is later copied into a directory under lib for

each of the build configurations which allows to have different

build options for different configurations too if you edit any

configuration-specific file.

Makefile parameters

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

When building using makefiles, you can specify many build settings

(unlike when using the project files where you are limited to choosing

just the configuration and platform). This can be done either by

passing the values as arguments when invoking make or by editing

build\msw\config.$(compiler) file where $(compiler) is same extension

as the makefile you use has (see below). The latter is good for

setting options that never change in your development process (e.g.

GCC\_VERSION or VENDOR). If you want to build several versions of

wxWidgets and use them side by side, the former method is better.

Settings in config.\* files are shared by all makefiles (including the

samples), but if you pass the options as arguments, you must use same

arguments you used for the library when building samples!

For example, to build the library in release mode you can either

change the "BUILD" variable definition in build\msw\config.$(compiler)

or use

> nmake -f makefile.vc BUILD=debug

> mingw32-make -f makefile.gcc BUILD=debug

depending on the compiler used.

The full list of the build settings follows:

BUILD=release

Builds release version of the library. It differs from default 'debug' in

lack of appended 'd' in name of library and uses the release CRT libraries

instead of debug ones. Notice that even release builds do include debug

information by default, see DEBUG\_FLAG for more information about it.

SHARED=1

Build shared libraries (DLLs). By default, DLLs are not built

(SHARED=0).

UNICODE=0

To completely disable Unicode support (default is UNICODE=1). It should not

be necessary to do this.

This option affect name of the library ('u' is appended in the default

Unicode build) and the directory where the library and setup.h are stored

(ditto).

WXUNIV=1

Build wxUniversal instead of native wxMSW

MONOLITHIC=1

Starting with version 2.5.1, wxWidgets has the ability to be built as

several smaller libraries instead of single big one as used to be the case

in 2.4 and older versions. This is called "multilib build" and is the

default behaviour of makefiles. You can still build single library

("monolithic build") by setting MONOLITHIC variable to 1.

USE\_GUI=0

Disable building GUI parts of the library, build only wxBase components used

by console applications. Note that if you leave USE\_GUI=1 then both wxBase

and GUI libraries are built.

USE\_$(LIBRARY)=0

Do not build the corresponding library (all libraries are built by

default). Library which can be disabled in this way are: AUI, HTML,

MEDIA, GL (the option name is USE\_OPENGL for this one), PROPGRID,

QA, RIBBON, RICHTEXT, STC, WEBVIEW, XRC.

RUNTIME\_LIBS=static

Links static version of C and C++ runtime libraries into the executable, so

that the program does not depend on DLLs provided with the compiler (e.g.

Visual C++'s msvcrt.dll or Borland's cc3250mt.dll).

Caution: Do not use static runtime libraries when building DLL (SHARED=1)!

DEBUG\_FLAG=0

DEBUG\_FLAG=1

DEBUG\_FLAG=2

Specifies the level of debug support in wxWidgets. Notice that

this is independent from both BUILD and DEBUG\_INFO options. By default

always set to 1 meaning that debug support is enabled: asserts are compiled

into the code (they are inactive by default in release builds of the

application but can be enabled), wxLogDebug() and wxLogTrace() are available

and \_\_WXDEBUG\_\_ is defined. Setting it to 0 completely disables all

debugging code in wxWidgets while setting it to 2 enables even the time

consuming assertions and checks which are deemed to be unsuitable for

production environment.

DEBUG\_INFO=0

DEBUG\_INFO=1

This option affects whether debugging information is generated. If

omitted or set to 'default' its value is determined the value of

the BUILD option.

DEBUG\_RUNTIME\_LIBS=0

DEBUG\_RUNTIME\_LIBS=1

(VC++ only.) If set to 1, msvcrtd.dll is used, if to 0, msvcrt.dll

is used. By default msvcrtd.dll is used only if the executable

contains debug info and msvcrt.dll if it doesn't. It is sometimes

desirable to build with debug info and still link against msvcrt.dll

(e.g. when you want to ship the app to customers and still have

usable .pdb files with debug information) and this setting makes it

possible.

TARGET\_CPU=X64|IA64

(VC++ only.) Set this variable to build for x86\_64 systems. If unset, x86

build is performed.

VENDOR=<your company name>

Set this to a short string identifying your company if you are planning to

distribute wxWidgets DLLs with your application. Default value is 'custom'.

This string is included as part of DLL name. wxWidgets DLLs contain compiler

name, version information and vendor name in them. For example

wxmsw311u\_core\_vc\_custom.dll is one of DLLs build using Visual C++ with

default settings. If you set VENDOR=mycorp, the name will change to

wxmsw311u\_core\_vc\_mycorp.dll.

CFG=<configuration name>

Sets configuration name so that you can have multiple wxWidgets builds with

different setup.h settings coexisting in same tree. The value of

this option is appended to the build directories names. This is

useful for building the library in some non-default configuration,

e.g. you could change wxUSE\_STL to 1 in %WXWIN%\include\wx\msw\setup.h and

then build with "CFG=-stl". Alternatively, you could build with e.g.

"RUNTIME\_LIBS=static CFG=-mt" when using MSVC.

COMPILER\_PREFIX=<string>

If you build with multiple versions of the same compiler, you can put

their outputs into directories like "vc6\_lib", "vc8\_lib" etc. instead of

"vc\_lib" by setting this variable to e.g. "vc6". This is merely a

convenience variable, you can achieve the same effect (but different

directory names) with the CFG option.

CFLAGS

CXXFLAGS

CPPFLAGS

LDFLAGS

Additional flags to be used with C compiler, C++ compiler, C

preprocessor (used for both C and C++ compilation) and linker,

respectively.

Building Applications Using wxWidgets

=====================================

NB: The makefiles and project files provided with wxWidgets samples show which

flags should be used when building applications using wxWidgets so in case

of a problem, e.g. if the instructions here are out of date, you can always

simply copy a makefile or project file from %WXWIN%\samples\minimal or some

other sample and adapt it to your application.

Independently of the compiler and make/IDE you are using you must do the

following to use wxWidgets sources under the directory $WXWIN (notice that

different tool chains refer to environment variables such as WXWIN in

different ways, e.g. MSVC users should use $(WXWIN) instead of just $WXWIN):

\* Add $WXWIN\include to the

- compiler

- resource compiler

include paths.

\* Define the following symbols for the preprocessor:

- \_\_WXMSW\_\_ to ensure you use the correct wxWidgets port.

- \_UNICODE unless you want to use deprecated ANSI build of wxWidgets.

- NDEBUG if you want to build in release mode, i.e. disable asserts.

- WXUSINGDLL if you are using DLL build of wxWidgets.

\* If using MSVC 7 only (i.e. not for later versions), also define

wxUSE\_RC\_MANIFEST=1 and WX\_CPU\_X86.

\* Add $WXWIN\lib\prefix\_lib-or-dll to the libraries path. The prefix depends

on the compiler, by default it is "vc" for MSVC, "gcc" for g++ and so on.

\* Add the list of libraries to link with to the linker input. The exact list

depends on which libraries you use and whether you built wxWidgets in

monolithic or default multlib mode and basically should include all the

relevant libraries from the directory above, e.g. "wxmsw31ud\_core.lib

wxbase31ud.lib wxtiffd.lib wxjpegd.lib wxpngd.lib wxzlibd.lib wxregexud.lib

wxexpatd.lib" for a debug build of an application using the core library only

(all wxWidgets applications use the base library).

Microsoft Visual C++ users can simplify the linker setup by prepending

"$(WXWIN)\include\msvc" to the include path (it must come before the

"$(WXWIN)\include" part!) and omitting the last step: the required libraries

will be linked in automatically using the "#pragma comment(lib)" feature of

this compiler.