Compiling - Windows

The officially supported Windows build platform for OpenCPN 4.1 beta and later is **Microsoft Visual Studio 2013** Express for Windows Desktop.
It is still possible to perform the build with older versions of Visual Studio - particularly if you are still running Windows XP on your development machine, you can not install a Visual Studio version newer than 2010. If what you have to do differs from the default VS2013 workflow, it is clearly noted in the instructions bellow.
Note: Older versions of OpenCPN, up to 4.0, used the VS2010 toolchain and wxWidgets 2.8, refer to the history of this page for the build instructions in case you need them.

Important note before you start

The order of the steps described below really matters.
Don't skip any steps not explicitly marked as optional and don't change their order unless you really know what you are doing.
It's an excellent idea to read the whole text first and make sure you understand what it's talking about, especially if you are new to software development.
If you encounter any problems, please get to us in the forum and tell us where you are failing so we can help you and improve these instructions for the others.

1. Prerequisites

1.1 Visual Studio 2013

Get **Visual Studio 2013 Express for Windows Desktop** or **Visual Studio 2013 Community Edition** (with Update 4) from **https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx**
If you use Windows XP or Vista, upgrade to Windows 7 or newer or get Visual Studio 2010 instead.

- Install Visual Studio - you can disable all the optional features offered to save some space

1.2 Git

Get the Git installation packages from **http://msysgit.github.io**

- Install and let it register in the **PATH** environment variable
- The defaults for all the installation settings are fine except the following:
    - On the **Adjusting your PATH environment**, select **Run Git from the Windows Command Prompt**.
    - On the **Choosing CR/LF behavior** select **Checkout as-is, Commit Unix LF**.
        - This is really important, the codebase uses Unix LF line endings and committing CR/LF makes the commits huge and hides their real content.
    - If your git is already installed, please edit the **.git/config** file in your local working copy of the OpenCPN source tree and set **autocrlf = input** in the **[core]** section on top.
    - If you want a tutorial, have a look at the series of articles starting at http://www.lostechies.com/blogs/jason_meridth/archive/2009/06/01/git-for-windows-developers-git-series-part-1.aspx

1.3 Cmake

Get the latest CMake installation packages from **http://www.cmake.org**

- Install and let it register in the **PATH** environment variable

1.4 POedit

Get the latest POedit installation package from **http://www.poedit.net**

- Install
- On 32bit Windows, add **C:\Program Files\Poedit\GettextTools\bin** to **PATH** environment variable. On a 64bit system, the path will be C:\Program Files (x86)\Poedit\GettextTools\bin

- On Windows 7 and later, open Computer, click System Properties, and in the left column click Advanced System Settings.
- On the Advanced tab, click on Environment Variables button and add the path in which msgfmt.exe resides to the PATH system variable.
- On Windows XP, right-click My Computer, select Properties...

1.5 NSIS

In case you want to create installation packages, get NSIS Unicode 2.46.5 from **http://www.scratchpaper.com/**

- Install
- Due to a "bug" in CMake, which only looks at "HKEY_LOCAL_MACHINE\SOFTWARE\NSIS" for the installation location of NSIS and the Unicode version adds its registry key in "HKEY_LOCAL_MACHINE\SOFTWARE\NSIS\Unicode", there is some registry tweaking needed.

Just copy the value (it's the installation path) of the "HKEY_LOCAL_MACHINE\SOFTWARE\NSIS\Unicode" key to "HKEY_LOCAL_MACHINE\SOFTWARE\NSIS".
Alternatively you can just run the batch file 'CopyNSISUnicodeRegKey.bat' which is also included in the GIT repository -> buildwin\NSIS_Unicode\CopyNSISUnicodeRegKey.bat
Depending on your security settings, mainly on Windows 7 and newer, you may have to run it as Administrator (right-click the file and select "Run as administrator".)
This means if you also want to use the ANSI NSIS version you first have to change the value of "HKEY_LOCAL_MACHINE\SOFTWARE\NSIS" registry key according to the installation path of the respective version you want to use.
To make the installer package use proper language name transaltions, it's necessary to modify file C:\Program Files\NSIS\Unicode\Contrib\Language files\Norwegian.nsh and change the line

```
!insertmacro LANGFILE "Norwegian" "Norwegian"
```

to

```
!insertmacro LANGFILE "Norwegian" "Norsk"
```

- You should also read the Modularized Packaging chapter of this manual.

2. wxWidgets 3

- Get the latest 3.0 release from **http://wxwidgets.org/downloads/** (at the time of this writing 3.0.2)
- Uncompress to your drive (we will assume to **C:\wxWidgets-3.0.2** in this text)

2.1 Compiling from the command line (recommended)

- Run the **Developer Command Prompt for VS2013**
- Go to your wxWidgets build tree (**cd C:\wxWidgets-3.0.2\build\msw**) and build both release and debug configurations, compatible with Windows XP

nmake -f makefile.vc BUILD=release SHARED=1 CFLAGS=/D_USING_V120_SDK71_ CXXFLAGS=/D_USING_V120_SDK71_

nmake -f makefile.vc BUILD=debug SHARED=1 CFLAGS=/D_USING_V120_SDK71_ CXXFLAGS=/D_USING_V120_SDK71_

In case you are using Visual Studio 2010, the build commands are:
nmake -f makefile.vc BUILD=release SHARED=1

nmake -f makefile.vc BUILD=debug SHARED=1

2.2 Compiling from Visual Studio IDE (optional)

This option is more work and not needed for 99% of people.

- In Visual Studio, open **wx_vc12.sln**

- Select all project from the ***Project Explorer***, right-click, select ***Properties***, Select ***All Configurations*** from the ***Configuration:*** dropdown on top and in ***Configuration Properties*** -> ***General*** set the ***Platform Toolset*** to **Visual Studio 2013 - Windows XP (v120_xp)**
- Build both the **Debug** and **Release** DLL targets
- In case you are using Visual Studio 2010, use the **wx_vc10.sln** solution and don't change the platform toolset

2.3 Add wxWidgets to your PATH

In order for Cmake to find wxWidgets, you must add your wxWidgets root director C:\${WXDIR} (for example C:\wxWidgets-3.0.2) to your PATH environment variable.
To be able to run debug builds and release builds *without install* add C:\${WXDIR}\lib\vc_dll to your PATH.
After doing this, you have to restart the running programs (cmd.exe, cmake-gui, VisualStudio etc) to make sure they "see" the changed environment variables.
If you are unsure, restart Windows and everything will be set. If you don't do it, you will have problems running your debug builds later.
If you have problems with cmake not finding your wxWidgets installation, try also creating another environment variable called WXWIN with a value of C:\${WXDIR} (for example C:\wxWidgets-3.0.2). Also, try creating an environment variable called wxWidgets_LIB_DIR=C:\${WXDIR}\lib\vc_dll and wxWidgets_ROOT_DIR=C:\${WXDIR}. Again, don't forget to restart the running programs involved in the build.

3. Get the OpenCPN source

Run **Developer Command Prompt for VS2013** from Start menu → Programs → Microsoft Visual Studio → Visual Studio Tools

- To get the source for the first time, issue
  git clone git://github.com/OpenCPN/OpenCPN.git
  In case of error messages like this one:
  `"error: unable to create file buildwin/NSIS_Unicode/CopyNSISUnicodeRegKey.bat (Permission denied)"`
  observed under Windows 8.1, run the command from an Administrator console
- To update the code you cloned before, cd into the source directory
  cd OpenCPN
- and issue
  git fetch --all

3.1 Get the binary dependencies

In case you want to make the Release builds and create the setup packages, you must get **OpenCPN_buildwin.7z** (In case you are using Visual Studio 2010, use **OpenCPN_buildwin-vc10.7z** instead) and extract the archive into your toplevel OpenCPN source directory created by the clone operation above. The archive contains some binary files needed to link OpenCPN and produce the installer. In case you just want to develop/debug OpenCPN, this step is not needed.
In case you need the PDB files for the prebuilt libraries (unlikely, really, if you don't know what for, you don't), get them from here.

4. Build OpenCPN

- Run **Developer Command Prompt for VS2013** from Start menu → Programs → Microsoft Visual Studio → Visual Studio Tools
- CD into your the topmost source directory and create a directory named **build** under it:

mkdir build

4.1a – Configuring the build from command line (recommended):

- cd into the **build** directory

- issue **cmake -T v120_xp ..**
- In case you are using Visual Studio 2010, the command is **cmake ..**

4.1b – Configuring the build Using Cmake-gui (in case the previous was too simple for you)

- Run "CMake (cmake-gui)" from Start menu → Programs → Cmake 3.2.
- Fill in your source and build directories.
- source = .../OpenCPN
- build = .../OpenCPN/build
- Click on the Configure button.
- If you are asked to choose the generator, select "Visual Studio 10".
- The information which appeared is red and the Generate button stays disabled? Just hit Configure again... Ignore GTK2_GTK_INCLUDE_DIR-NOTFOUND and wxwidgets_wxrc_EXECUTABLE_NOTFOUND.
- Click on the Generate button.
- Solution and project files should be created in your build directory.

IMPORTANT suggestion:
Use CMAKE GUI tool to configure OpenCPN to verify that wxWidgets_LIB_DIR points to the {root}/lib/vc_dll directory.
This check is necessary since the cmake FindWxWidgets module sometimes finds the wrong source and/or build config.

If you are using CMake version 3.0 or later you will get warnings about Policy CMP0043.
These can be ignored.

4.2a – Compiling from the command line

- Run **Developer Command Prompt for VS2013** from Start menu → Programs → Microsoft Visual Studio → Visual Studio Tools.
- cd into the **build** directory.
- issue for a release build
  cmake --build . --config release
  or for a debug build
  cmake --build . --config debug
  Note that if you don't use the --config parameter, a debug build is performed
- Wait for the build to complete.

4.2b – Compiling from Visual Studio

- Open the solution created by cmake (build/OpenCPN.sln).
- Compile the whole solution or individual projects.
- You have to compile project opencpn before you can compile any plugins (to be fixed in the configuration process)

If you want to debug, don't forget to select opencpn as a start-up project and if you didn't add the WX DLL path to the PATH environment variable earlier, copy the needed WX DLLs to the build directory (Debug or Release, depending on which version you build). The DLLs can be found in C:\${WXDIR}\lib\vc_dll and you will need:
Debug: wxbase30ud_net_vc_custom.dll, wxbase30ud_vc_custom.dll, wxbase30ud_xml_vc_custom.dll, wxmsw30ud_adv_vc_custom.dll, wxmsw30ud_core_vc_custom.dll
Release: wxbase30u_net_vc_custom.dll, wxbase30u_vc_custom.dll, wxbase30u_xml_vc_custom.dll, wxmsw30u_adv_vc_custom.dll, wxmsw30u_core_vc_custom.dll
See **6 - Running** below to prepare to run the Debug or Release build (from Visual Studio or otheriwse) without installing.

5 – Optional: Creating the installer package

Build the **PACKAGE** project and **opencpn_X.Y.Z_setup.exe** is created in your build directory (replace X with the release and Y with the build number). Use the following command:

cmake --build . --target package --config release
This will create a directory called "_CPack_Packages" and under release directory. You should find your install package under the NSIS sub-directory.

Currently the installer packs the DLLs from the git repository into the package. You have to replace them with your custom built DLLs after the installation if you want to experiment with different versions and build settings of the wxWidgets libraries.

6 – Running OpenCPN from the build directory (without installing)

- Create a folder named **uidata**
  - in the case of a ***Debug build*** in your **build** directory
  - in the case of a ***Release build*** in the **build/Release directory**
- and copy the following files from **src/bitmaps** into it:

styles.xml, toolicons_traditional.png, toolicons_journeyman.png, toolicons_journeyman_flat.png.

- Copy the following folders from the **data** subfolder of the source tree to your **build/Debug** folder (***Debug build***) or to the **build/Release** directory (***Release build***)

gshhs, **s57data**, **tcdata**.

- For a ***Release build*** you also need to copy the following from **builwin/crashrpt** to **build/Release**:

CrashRpt1402.dll, CrashSender1402.exe, crashrpt_lang.ini, dbghelp.dll

Running for the first time

To run the first time issue the following command from a command prompt in the **build/Debug** or **build/Release** directory:
opencpn.exe /p
This will generate an opencpn.ini file in the current directory as well as create the opencpn.log file.

7 – Something does not work as expected?

Before getting desperate, **read you openpcn.log logfile**, it is likely that the problem is clearly identified there.

‹ Compiling - Mac OS XCompiling Standalone Plugins and building Install Packages ›

- Printer-friendly version