# Building QT Apps in WebAssembly

(March 2021)

## What Works

Building QT apps manually via Linux Terminal seems to be the best option for making QT apps work with Wasm.

This process has been verified to work on both:

- A windows 10 computer using virtualbox to run Ubuntu 20.04
- A window 10 computer using WSL2 to run Ubuntu 20.04

---

# Necessary Dependencies (Only need to run these once)

1) sudo apt-get build-dep --yes qt5-default
2) sudo apt-get install --yes libxcb-xinerama0-dev git python cmake default-jre g++

# Install Emscripten

# Note:
emsdk activate && source emsdk_env.sh, need to be run per session since they are setting temporary environment variables. There is no harm in running these commands multiple times. If you get a message like " Cannot run target compiler 'em++'" it is likely because you have to update the environment variables for that session.

1) git clone https://github.com/juj/emsdk.git
2) cd emsdk
3) ./emsdk install latest
4) ./emsdk activate latest
5) source emsdk_env.sh
6) cd ..

# Build Qt (Only need to run these once)

#Note:
./configure && make will take a while

1) git clone -b 5.15.2 https://code.qt.io/qt/qt5.git
2) cd qt5
3) ./init-repository -f --module-subset=qtbase,qtdeclarative,qtwebsockets,qtsvg,qtquickcontrols,qtquickcontrols2,qtgraphicaleffects
4) ./configure -opensource -confirm-license -xplatform wasm-emscripten -release -static -no-feature-thread -nomake examples -no-dbus -no-ssl

5) make
6) cd ..

# Build your project (Project Specific)
#Note:
Some of the examples within this library of qt-webassembly-examples use QT quick library functionality. I could not get this library to work.

1) git clone https://github.com/msorvig/qt-webassembly-examples.git
2) cd qt-webassembly-examples/widgets_wiggly
3) ../../qt5/qtbase/bin/qmake
4) make
5) python -mSimpleHTTPServer
6) Open browser and go to 0.0.0.0:8000/widgets_wiggly.html

# What Doesn't Work

Official Qt Documentation on setting up Qt + Wasm <- **This garbage doesn't work**

## Various things I've learned: (**following official documentation**)

1) QT + Wasm doesn't have any good video tutorials. You are reliant pretty much entirely on official documentation, stackoverflow, and the Qt forums for assistance.
2) Attempting to get QT + Wasm working on Ubuntu is harder than windows when working with QtCreator. Made much more progress on windows.
    a) Clicking on QtCreator App on Ubuntu doesn't work. Must be run manually via terminal (not a big deal just something to know).
    b) QtCreator error messages are very unhelpful on Linux I found.
    c) Unable to get past "Invalid Toolchain error" on Qt Wasm Kit, Qt won't provide any details whatsoever about the issue. Had no issues with this on Windows.
3) If you install emscripten exactly according to the download documentation on the emscripten website QT will not find it natively.
    a) QT fails to both:
        i) Look in the correct directory. It looks in the home directory instead of the emsdk directory.
        ii) Resolve the variable (emsdk_path) in the .emscripten file (the file containing the pathing information).
    b) To get QT to recognize emscripten you have to install emscripten according to the download documentation.

i) Then at some point after running emsdk activate {version #} you must manually copy the .emscripten file that was generated to the home directory.

ii) You then must open that .emscripten file and manually configure the paths without the “emsdk_path +” prefix.

(1) Before

```
import os
emsdk_path = os.path.dirname(os.environ.get('EM_CONFIG')).replace('\\', '/')
NODE_JS = emsdk_path + '/node/14.15.5_64bit/bin/node.exe'
PYTHON = emsdk_path + '/python/3.7.4-pywin32_64bit/python.exe'
JAVA = emsdk_path + '/java/8.152_64bit/bin/java.exe'
LLVM_ROOT = emsdk_path + '/upstream/bin'
BINARYEN_ROOT = emsdk_path + '/upstream'
EMSCRIPTEN_ROOT = emsdk_path + '/upstream/emscripten'
TEMP_DIR = emsdk_path + '/tmp'
COMPILER_ENGINE = NODE_JS
JS_ENGINES = [NODE_JS]
```

(2) After

```
import os

NODE_JS = 'C:/Users/Kevin/emsdk/node/14.15.5_64bit/bin/node.exe'
PYTHON = 'C:/Users/Kevin/emsdk/python/3.7.4-pywin32_64bit/python.exe'
JAVA = 'C:/Users/Kevin/emsdk/java/8.152_64bit/bin/java.exe'
LLVM_ROOT = 'C:/Users/Kevin/emsdk/upstream/bin'
BINARYEN_ROOT = 'C:/Users/Kevin/emsdk/upstream'
EMSCRIPTEN_ROOT = 'C:/Users/Kevin/emsdk/upstream/emscripten'
TEMP_DIR = 'C:/Users/Kevin/emsdk/tmp'
COMPILER_ENGINE = NODE_JS
JS_ENGINES = [NODE_JS]
```

iii) QT should now find emscripten. You may need to click redetect compilers.

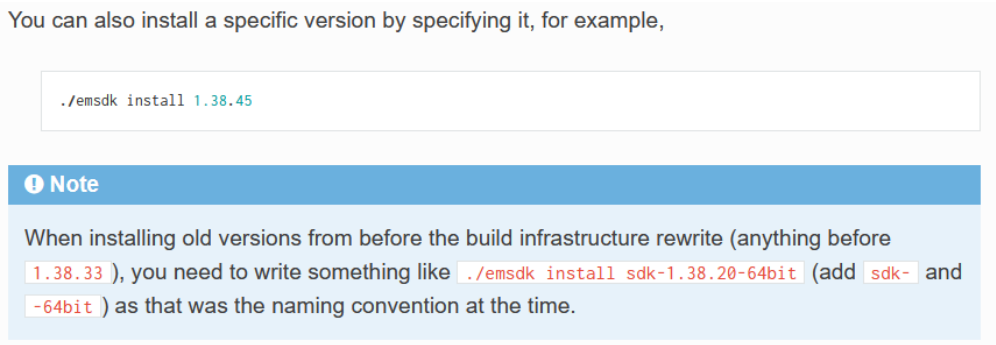
iv) You only need to do this once provided you don’t install a different/new emscripten version.

c) emsdk activate {version #} and emsdk_env.bat **must** be run every session prior to doing anything. Otherwise QtCreator will provide this error message:

i) Error while parsing file whatever.pro. Giving up.
Project ERROR: Cannot run target compiler 'em++'. Output:
==========
==========
Maybe you forgot to setup the environment?

4) On Windows you must install MinGW and the SED stream editor
   a) You must add the pathing information for these installs to your QtCreator kit it should look something like this:
      i) PATH=C:\Qt\Tools\mingw810_64\bin;C:\Program Files (x86)\GnuWin32\bin;${PATH}
5) When you try and use Qt + Wasm at some point during the build process it will fail to handle directories with spaces (yes really). I believe it is an issue with emscripten in particular.

    a) Found this out personally when I was trying to compile everything in a folder called ‘Senior Project’ and the build process was creating a ‘Senior’ file on my desktop and then immediately failing.
    b) So if you are trying this on windows use underscores no spaces.
6) By default Qt will try to build in the same directory as the source file directory. For example when building Senior_Project\WasmTest it will put the build directory next to it as Senior_Project\Build_of_WasmTest. This works perfectly fine for literally everything except for emscripten for some reason. You need to manually set the build directory as a completely independent external path.
7) If you try to build with emscripten repeatedly you must delete the previous build field first or it will just fail.
8) Occasionally if you install & uninstall different Qt versions a whole bunch for some reason QtCreator will sometimes refuse to boot even after a clean install.
    a) This is because both Qt as well as QtCreator (even after being cleanly uninstalled using their own uninstall tool) will leave .ini,metadata, and other small configuration files in the appdata directory that get screwed up jumping between versions.
    b) If this issue occurs where after a clean install you can’t get QtCreator boot you need to delete this directory **C:\Users\Kevin\AppData\Roaming\QtProject**
9) If you try to install older emscripten versions in an attempt to get an older combination of Qt + emscripten working. The official documentation on the emscripten website for installing old versions is flat out wrong.
    a) If you literally run the exact command **emsdk install sdk-1.38.20-64bit** it won’t work. The naming convention has since been changed and isn’t reflected on the official download documentation (Cool!).


You can also install a specific version by specifying it, for example,

```
./emsdk install 1.38.45
```

**Note**

When installing old versions from before the build infrastructure rewrite (anything before `1.38.33`), you need to write something like `./emsdk install sdk-1.38.20-64bit` (add `sdk-` and `-64bit`) as that was the naming convention at the time.

## Setting up Process:

1) Make a Qt Account (it is free).
2) Download the Qt Maintenance tool off the Qt website.
3) Install

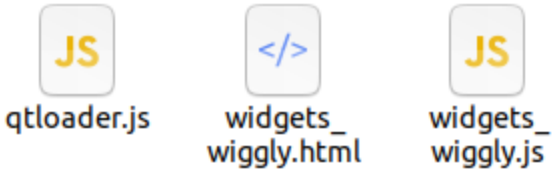
    a) A Qt version that has Wasm support
    b) QtCreator
    c) MinGW + Wasm only in the dropdown
    d) Should be less than 5 gigs if larger you are installing too much
4) Download or make sure you have MinGW
5) Download SED
6) Download and setup emscripten
    a) **git clone https://github.com/emscripten-core/emsdk.git** in the home directory
    b) Cd emsdk
    **c) Emsdk install {version #}**
    **d) Emsdk activate {version #}**
    **e) Emsdk_env.bat**
    f) The last 2 commands must be run every session.
    g) Copy the .emscripten file in the \emsdk directory to the home directory
    h) Edit the paths as detailed above and save the file.
7) Boot QtCreator
8) Add the MinGW and SED paths to the environmental variables of the wasm kit as detailed above.
9) Go to Tools -> Options -> Build & Run and change the build & run directory to a different folder than where the src files are going to be.

# QT + Wasm Runtime

So my understanding is there are 3 required runtime files
At least when you are running the compiled qt application on a browser using

*python3 -mhttp.server*

- qtloader.js
- appname.html
- appname.js




The appname.html is hosted locally in my case I have to go to 0.0.0.0:8000. It is a pretty standard boilerplate html file outside of the <script> stuff setting up QT.
That html file is calling qtloader.js.

```
                canvas.style.display = 'block';
            },
        });
        qtLoader.loadEmscriptenModule("widgets_wiggly");
    }
  </script>
  <script type="text/javascript" src="qtloader.js"></script>
</body>
```

Qtloader.js is essentially bootstrapping everything. This handles all the qt public api and qt library stuff behind the scenes. I believe this file is the same no matter what your application happens to be.

When it is done it then calls appname.js.

```
        // Fetch emscripten generated javascript runtime
        var emscriptenModuleSource = undefined
        var emscriptenModuleSourcePromise = fetchText(applicationName +
".js").then(function(source) {
            emscriptenModuleSource = source
        });
```

appname.js is difficult to read by a human since it does not have any line breaks at all. It is a 1/4 MB text file all on a single line. The reason for this is the appname.js file is the .wasm compiled webassembly converted into a javascript file.

So my understanding is that the appname.wasm is the webassembly byte code and the appname.js is the javascript interpretation of that bytecode.

Based on all of this I believe that if you could import qtloader.js into intellij the app should run. Since all it is doing is calling another javascript file in the same directory appname.js which is holding all the qt application data. Albeit you may have to resolve some variable names manually. Otherwise perhaps it is possible to just skip qtloader.js and add in appname.js directly idk.


HEY LADIES
There is no place like
Are you a cross-platform application framework?
Because you look like a Qt.