

# Mac OS X Build Instructions and Notes

---

This guide will show you how to build dynamicd (headless client) for OSX.

## Notes

---

- Tested on OS X 10.7 through 10.13.6 on 64-bit Intel processors only.
- All of the commands should be executed in a Terminal application. The built-in one is located in `/Applications/Utilities`.

## Preparation

---

You need to install XCode with all the options checked so that the compiler and everything is available in `/usr` not just `/Developer`. XCode should be available on your OS X installation media, but if not, you can get the current version from <https://developer.apple.com/xcode/>. If you install Xcode 4.3 or later, you'll need to install its command line tools. This can be done in `Xcode > Preferences > Downloads > Components` and generally must be re-done or updated every time Xcode is updated.

There's also an assumption that you already have `git` installed. If not, it's the path of least resistance to install [Github for Mac](#) (OS X 10.7+) or [Git for OS X](#). It is also available via Homebrew.

You will also need to install [Homebrew](#) in order to install library dependencies.

The installation of the actual dependencies is covered in the Instructions sections below.

# Instructions: Homebrew

---

## Install dependencies using Homebrew for Daemon & Qt

```
$ brew install git autoconf automake libevent libtool boost  
--c++11 miniupnpc openssl pkg-config protobuf --c++11  
qt berkeley-db4
```

Because of OS X having LibreSSL installed we have to tell the compiler where OpenSSL is located:

```
$ export LDFLAGS=-L/usr/local/opt/openssl/lib  
$ export CPPFLAGS=-I/usr/local/opt/openssl/include
```

or you can instead symlink your newly installed OpenSSL:

```
$ sudo ln -s openssl-1.0.2q /usr/local/openssl
```

(the above version of OpenSSL may differ to the one you have installed, amend to suit)

After exiting you will want to symlink berkeley-db4 and qt:

```
$ brew link berkeley-db4 --force  
$ brew link qt --force  
$ brew link boost --c++11 --force
```

## Disable-wallet mode

---

When the intention is to run only a P2P node without a wallet, dynamic may be compiled in disable-wallet mode with:

```
./configure --disable-wallet
```

In this case there is no dependency on Berkeley DB 4.8.

Mining is also possible in disable-wallet mode, but only using the `getblock` template RPC call not `getwork` .

## AVX2 Mining Optimisations

---

For increased performance when mining, AVX2 optimisations can be enabled.

At configure time:

```
--enable-avx2
```

CPU's with AVX2 support:

```
Intel
  Haswell processor, Q2 2013
  Haswell E processor, Q3 2014
  Broadwell processor, Q4 2014
  Broadwell E processor, Q3 2016
  Skylake processor, Q3 2015
  Kaby Lake processor, Q3 2016
  (ULV mobile)/Q1 2017(desktop/mobile)
  Coffee Lake processor, expected in 2017
  Cannonlake processor, expected in 2018
AMD
  Carrizo processor, Q2 2015
  Ryzen processor, Q1 2017
```

## AVX512 Mining Optimisations

---

For increased performance when mining, AVX512 optimisations can be enabled.

At configure time:

```
--enable-avx512f
```

CPU's with AVX512 support:

#### Intel

Xeon Phi x200/Knights Landing processor, 2016  
Knights Mill processor, 2017  
Skylake-SP processor, 2017  
Skylake-X processor, 2017  
Cannonlake processor, expected in 2018  
Ice Lake processor, expected in 2018

## GPU Mining

---

To enable GPU mining within the wallet, OpenCL or CUDA can be utilised. For CUDA to be utilised please use CUDA 9.1 or newer and ensure you have graphics drivers installed. OSX contains libraries and headers for OpenCL and no dependencies are required to be downloaded.

For CUDA please visit: <https://docs.nvidia.com/cuda/cuda-installation-guide-mac-os-x/index.html>

At configure time for OpenCL(Nvidia/AMD):

```
--enable-gpu
```

At configure time for CUDA(Nvidia):

```
--enable-gpu --enable-cuda
```

## Building Dynamic

---

1. Clone the github tree to get the source code and go into the directory.

```
git clone https://github.com/duality-solutions/dynamic.git
cd dynamic
```

2. Build dynamicd:

```
./autogen.sh
./configure
make
```

3. It is also a good idea to build and run the unit tests:

```
make check
```

4. (Optional) You can also install `dynamicd` to your path:

```
make install
```

## Use Qt Creator as IDE

---

You can use Qt Creator as IDE, for debugging and for manipulating forms, etc.

Download Qt Creator from <http://www.qt.io/download/>. Download the "community edition" and only install Qt Creator (uncheck the rest during the installation process).

1. Make sure you installed everything through homebrew mentioned above
2. Do a proper `./configure --with-gui=qt5 --enable-debug`
3. In Qt Creator do "New Project" -> Import Project -> Import Existing Project
4. Enter "dynamic-qt" as project name, enter `src/qt` as location
5. Leave the file selection as it is
6. Confirm the "summary page"
7. In the "Projects" tab select "Manage Kits..."
8. Select the default "Desktop" kit and select "Clang (x86 64bit in /usr/bin)" as compiler
9. Select LLDB as debugger (you might need to set the path to your installation)
10. Start debugging with Qt Creator

## Creating a release build

---

You can ignore this section if you are building `dynamicd` for your own use.

`dynamicd/dynamic-cli` binaries are not included in the `Dynamic-Qt.app` bundle.

If you are building `dynamicd` or `Dynamic-Qt` for others, your build machine should be set up as follows for maximum compatibility:

All dependencies should be compiled with these flags:

```
-mmacosx-version-min=10.7  
-arch x86_64  
-isysroot $(xcode-select --print-path)/Platforms/MacOSX.platform/  
Developer/SDKs/MacOSX10.7.sdk
```

Once dependencies are compiled, see `release-process.md` for how the `Dynamic-Qt.app` bundle is packaged and signed to create the `.dmg` disk image that is distributed.

## Running

---

It's now available at `./dynamicd`, provided that you are still in the `src` directory. We have to first create the RPC configuration file, though.

Run `./dynamicd` to get the filename where it should be put, or just try these commands:

```
echo -e "rpcuser=dynamicrpc\nrpcpassword=$(xxd -l 16 -p  
/dev/urandom)" > "/Users/${USER}/Library/  
Application Support/Dynamic/dynamic.conf"  
  
chmod 600 "/Users/${USER}/Library/Application Support/  
Dynamic/dynamic.conf"
```

The next time you run it, it will start downloading the blockchain, but it won't output anything while it's doing this. This process may take several hours; you can monitor its process by looking at the `debug.log` file, like this:

```
tail -f $HOME/Library/Application\ Support/Dynamic/debug.log
```

## Other commands:

---

```
./dynamicd -daemon # to start the dynamic daemon.  
./dynamic-cli --help # for a list of command-line options.  
./dynamic-cli help    # When the daemon is running,  
to get a list of RPC commands
```