

UNIX BUILD NOTES

Some notes on how to build [Dynamic](#) in Unix.

Note

Always use absolute paths to configure and compile Dynamic and the dependencies,
for example, when specifying the the path of the dependency:

```
../dist/configure --enable-cxx --disable-shared --with-pic --pre
```

Here BDB_PREFIX must absolute path - it is defined using \$(pwd) which ensures
the usage of the absolute path.

To Build

```
./autogen.sh  
./configure  
make  
make install # optional
```

This will build dynamic-qt as well if the dependencies are met.

Dependencies

These dependencies are required:

Library	Purpose	Description
libssl	SSL Support	Secure communications
libboost	Boost	C++ Library
libevent	Networking	OS independent asynchronous networking

Optional dependencies:

Library	Purpose	Description
miniupnpc	UPnP Support	Firewall-jumping support
libdb4.8	Berkeley DB	Wallet storage (only needed when wallet enabled)
qt	GUI	GUI toolkit (only needed when GUI enabled)
protobuf	Payments in GUI	Data interchange format used for payment protocol (only needed when GUI enabled)
libqrencode	QR codes in GUI	Optional for generating QR codes (only needed when GUI enabled)
libzmq3	ZMQ notification	Optional, allows generating ZMQ notifications (requires ZMQ version >= 4.x)

For the versions used in the release, see [release-process.md](#) under *Fetch and build inputs*.

System requirements

C++ compilers are memory-hungry. It is recommended to have at least 3 GB of memory available when compiling Dynamic.

Dependency Build Instructions: Ubuntu & Debian

It is required to build Dynamic on Ubuntu 18.04LTS(Bionic) or later due to C++14/GCC7 requirements. Also OpenSSL 1.1.0g is included in Ubuntu 18.04LTS and later, however it is suggested to use OpenSSL 1.1.1LTS.

Build requirements:

```
sudo apt-get install build-essential libtool autotools-dev autoc
```

For Ubuntu 18.04LTS(Bionic) and later, or Debian 7 and later; libboost-all-dev has to be installed:

```
sudo apt-get install libboost-all-dev
```

db4.8 packages are available [here](#). You can add the repository using the following command:

```
sudo add-apt-repository ppa:bitcoin/bitcoin  
sudo apt-get update
```

Ubuntu 18.04 and later have packages for libdb 5.3.21 but using these will break binary wallet compatibility, and is not recommended.

for Debian 7 (Wheezy) and later:

The oldstable repository contains db4.8 packages.

Add the following line to /etc/apt/sources.list, replacing [mirror] with any official debian mirror.

```
deb http://[mirror]/debian/ oldstable main
```

To enable the change run

```
sudo apt-get update
```

for other Debian & Ubuntu (with ppa):

```
sudo apt-get install libdb4.8-dev libdb4.8+-dev
```

Optional (see --with-miniupnpc and --enable-upnp-default):

```
sudo apt-get install libminiupnpc-dev
```

ZMQ dependencies (provides ZMQ API 4.x):

```
sudo apt-get install libzmq3-dev
```

Dependencies for the GUI: Ubuntu & Debian

If you want to build Dynamic-Qt, make sure that the required packages for Qt development are installed. Qt 5 is necessary to build the GUI. If both Qt 4 and Qt 5 are installed, Qt 5 will be used. Pass `--with-gui=qt5` to `configure` to choose Qt5. To build without GUI pass `--without-gui`.

For Qt 5 you need the following:

```
sudo apt-get install libqt5gui5 libqt5core5a libqt5dbus5 qttools
```

libqrencode (optional) can be installed with:

```
sudo apt-get install libqrencode-dev
```

Once these are installed, they will be found by `configure` and a `dynamic-qt` executable will be built by default.

Notes

The release is built with GCC and then "strip dynamicd" to strip the debug symbols, which reduces the executable size by about 90%.

miniupnpc

The dependencies for miniupnpc are included above within the 'Dependency Build Instructions: Ubuntu & Debian' section, however, for any reason you wish to build the source, please follow these instructions.

[miniupnpc](#) may be used for UPnP port mapping. It can be downloaded from [here](#). UPnP support is compiled in and turned off by default. See the `configure` options for upnp behavior desired:

<code>--without-miniupnpc</code>	No UPnP support miniupnp not required
<code>--disable-upnp-default</code>	(the default) UPnP support turned off by default
<code>--enable-upnp-default</code>	UPnP support turned on by default at runtime

To build:

```
tar -xzvf miniupnpc-1.6.tar.gz
cd miniupnpc-1.6
make
sudo su
make install
```

Berkeley DB

It is recommended to use Berkeley DB 4.8 and is included in the dependencies above in the 'Dependency Build Instructions: Ubuntu & Debian' section.

If you have to, or wish to build Berkeley DB 4.8 yourself:

```
bash
DYNAMIC_ROOT=$(pwd)

# Pick some path to install BDB to, here we create a directory
BDB_PREFIX="${DYNAMIC_ROOT}/db4"
mkdir -p $BDB_PREFIX

# Fetch the source and verify that it is not tampered with
wget 'http://download.oracle.com/berkeley-db/db-4.8.30.NC.tar.gz'
echo '12edc0df75bf9abd7f82f821795bcee50f42cb2e5f76a6a281b857'
# -> db-4.8.30.NC.tar.gz: OK
tar -xzvf db-4.8.30.NC.tar.gz

# Build the library and install to our prefix
cd db-4.8.30.NC/build_unix/
# Note: Do a static build so that it can be embedded into the wallet
../dist/configure --prefix=/usr/local --enable-cxx
make
sudo make install

# Configure Dynamic to use our own-built instance of BDB
cd $DYNAMIC_ROOT
./configure (other args...) LDFLAGS="-L${BDB_PREFIX}/lib/" C
```

Note: You only need Berkeley DB if the wallet is enabled (see the section *Disabling-Wallet mode* below).

Boost

If you need to build Boost yourself, in terminal enter:

```
sudo su
./bootstrap.sh
./bjam install
```

Security

To help make your Dynamic installation more secure by making certain attacks impossible to exploit even if a vulnerability is found, binaries are hardened by default.

This can be disabled with:

Hardening Flags:

```
./configure --enable-hardening
./configure --disable-hardening
```

Hardening enables the following features:

- Position Independent Executable
Build position independent code to take advantage of Address Space Layout Randomization offered by some kernels. An attacker who is able to cause execution of code at an arbitrary memory location is thwarted if he doesn't know where anything useful is located.
The stack and heap are randomly located by default but this allows the code section to be randomly located as well.

On an Amd64 processor where a library was not compiled with -fPIC, this will cause an error such as: "relocation R_X86_64_32 against `.....' can not be used when making a shared object;"

To test that you have built PIE executable, install `scanelf`, part of `paxutils`, and use:

```
scanelf -e ./dynamicd
```

The output should contain:

```
TYPE
ET_DYN
```

- **Non-executable Stack**

If the stack is executable then trivial stack based buffer overflow exploits are possible if vulnerable buffers are found. By default, `dynamic` should be built with a non-executable stack but if one of the libraries it uses asks for an executable stack or someone makes a mistake and uses a compiler extension which requires an executable stack, it will silently build an executable without the non-executable stack protection.

To verify that the stack is non-executable after compiling use:

```
scanelf -e ./dynamicd
```

the output should contain:

```
STK/REL/PTL
RW- R-- RW-
```

The `STK RW-` means that the stack is readable and writeable but not executable.

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/mob
- 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. Please use GCC/G++ 6.4 or newer and for CUDA to be utilised please use CUDA 9.1 or newer and ensure you have graphics drivers installed.

For OpenCL you need the following:

```
sudo apt-get install ocl-icd-ocl-dev
```

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

At configure time for OpenCL(Nvidia/AMD):

```
--enable-gpu
```

At configure time for CUDA(Nvidia):

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

Example Build Command

Qt Wallet and Deamon, CLI version build without GPU support:

```
./autogen.sh && ./configure --with-gui --disable-gpu && make
```

CLI and Deamon Only build without GPU support:

```
./autogen.sh && ./configure --without-gui --disable-gpu && make
```

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