# Installing and configuring qOp on OS X

Version 2.5.0.1 (October 2017)

This document describes how to install, on the OS X operating system, the qOp package and certain software packages that are pre-requisites for it. If you do not have the XCode build environment for C/C++ programs installed, please see the appendices.

This document uses the following typeface conventions:

- Commands for you to enter are shown in a fixed-width font, such as: `ps -ef`
- File and directory names are shown in a bold fixed-width font, such as: **`foo.bar`**
- The shell prompt is indicated via `$` and so the `cp` command would appear as: `$ cp`

## 1. Installing qOp from the installation file

The qOp installation file should have been provided to you. The installation file is named:

**`qOp.osx_2.5.tar.gz`**

### A. Decide whether to do a single- or multi-user installation

The qOp package may be installed for a single user (typically on a laptop or desktop system) or for multiple users (typically on a shared server). For a single user, it is often simplest to install it in your home directory. For multiple users, `/opt/local/` and `/usr/local/` are common choices of install directory. This document assumes that you choose a directory to which we append "`/qOp`" and refer to it as the installation directory or `DWAVE_HOME` (later in this document that value will be stored in the environment variable `DWAVE_HOME`).

### B. Move the qOp installation file into the installation directory

Move the qOp installation file into the installation directory.

### C. Confirm the contents of the installation file

If you have reason to believe the download or transfer of the qOp tar-file may have introduced errors, you can confirm that it was downloaded correctly with the following steps.

Start a OS X shell window (typically via the Terminal application) and examine the installation file as follows:

```
$ shasum -c qOp.osx_2.5.tarGzShasum.txt
```

- Response should be
  `> qOp.osx_2.5.tar.gz: OK`
- If not an OK response, there was a problem with the download

```
$ gunzip qOp.osx_2.5.tar.gz
```

```
$ shasum -c qOp.osx_2.5.tar.shasum
```

- Response should be
  `> qOp.osx_2.5.tar: OK`
- If not an OK response, there was a problem with the unzip

Note that the gunzip invocation above is redundant with the gunzip in the next section.

## D. Open the qOp installation file and install the software

Start a Unix shell window and do the following:

```
$ gunzip qOp.osx_2.5.tar.gz
```

This should return to your shell prompt, with no error. This step converts the file from a "GZIPped TAR file" into a "TAR file." (The original file was compressed so that it required less space. A TAR file is an archive file, allowing storing files in a "directory" structure.")

```
$ tar xvf qOp.osx_2.5.tar
```

The screen will print several lines of text quickly. This "untars" the "TAR file", creating the desired directory structure, and installs the qOp software. This will create a directory named qOp in the installation directory.

## E. Inspect the qOp software structure

In your OS X Terminal shell window, type

```
$ cd qOp
```

which means "change directory", and moves you into the `qOp` directory. Then, do

```
$ ls -als
```

which is the Unix command `ls`, for "list", which shows the files in the directory. The "-als" options cause the list to include "dot files" (those starting with "`.`", which are normally omitted by `ls`) and makes the output more readable.

Here are the top-level contents of the `qOp` directory:

| Name | Type | Contents |
|---|---|---|
| `.dwrc` | Initialization file | Connection information for the local simulator |
| `Quantum Apprentice.xlsm` | Excel Spreadsheet | Visualization and simulation tool |
| `bin` | Directory | Executables: `toq`, `dw-exec`, `qbsolv`, `qsage`, etc. |
| `doc` | Directory | Documents for C API, man pages for utilities |
| `dwave_qbsolv` | Python package | Enables Python calls to qbsolv |
| `dwave_sapi.h` | C header file | |
| `epqmi.h` | C header file | For use by programs calling functions in libepqmi.a |
| `examples` | Directory | Example source code for C API, ToQ, dw, qsage, and qbsolv/QUBOs |
| `libdwave_sapi.dylib` | C dynamic library | SAPI 2.6 library |
| `libepqmi.a` | C dynamic library | Library version of dw functionality |

| | | |
|---|---|---|
| licenses | Directory | Licenses |
| qOp-version.txt | Text file | Version information |
| qOp_2.5_Release_Note.txt | Text file | Information about the qOp 2.5 release |
| qOp_osx_INSTALL.pdf | PDF file | This document |
| src | Directory | Source code, namely of qbsolv, for building dwave_qbsolv |
| workspace.c4<br>workspace.dw2x_vfyc<br>workspace.dw2000q_vfyc | Workspace directories | QMI and SOL files for the 128-qubit simulator, and the 1152-qubit and 2048-qubit virtual full yield solvers |

# 2. Configuring the system for qOp

## A. [Only for El Capitan or later] Install the SAPI library into the system directory

If you are running OS X El Capitan (v. 10.11) or later release, you need to copy the SAPI library into a protected library directory known to the system. For this you will need to use the `sudo` command ("super-user do"), which executes the indicated command as though you were super-user, for which you will need permissions on your system.

```
$ sudo cp libdwave_sapi.dylib /usr/local/lib/
```

If you are not running El Capitan, skip this step.

## B. Edit your `~/.bash_profile` and `~/.bashrc` files

Next you will need to edit your **`~/.bash_profile`** and **`~/.bashrc`** files to include configuration information for the qOp package. In your OS X shell session, change to your home directory via:

```
$ cd
```

Append the following lines to your `~/.bashrc` file, using **`vi`** or any text editor you prefer:

```
export DWAVE_HOME=$HOME/qOp                # see note below
export DWAVE_HOME=<multiuserInstallDirectory>/qOp
export DWAVE_WORKSPACE=$HOME/qOp           # see note below
export PATH=$PATH:$DWAVE_HOME/bin
export DYLD_LIBRARY_PATH=$DWAVE_HOME       # see note below
export BASH_ENV=$DWAVE_HOME/bin/dw_setup   # see note below
. $BASH_ENV
```

Note: You should include only one of the two green lines that set the DWAVE_HOME environment variable. If you're installing qOp as a single-user installation in your home directory, use the first line. If qOp was installed for multiple users, you will need to find the name of the directory where it was installed (i.e., replace "`<multiuserInstallDirectory>`" with the actual directory name).

Note: If you are running a multi-user qOp installation, setting `DWAVE_WORKSPACE` is the way to control where in your file structure qOp will store temporary and configuration files. If you are running a single-user qOp installation, you may omit setting `DWAVE_WORKSPACE`.

Note: If you are not running El Capitan or later, you need the third line above (in red), in place of the `sudo` command. If you are running El Capitan or later, you do not need the third line above and should omit it.

Note: The name of the **`dw`** set-up script changed between qOp 2.3.1 and 2.4. To avoid confusion with the **`dw`** shell command, the set-up script is now called **`dw_setup`**. If you are upgrading from qOp 2.3.1 or earlier release, you will need to change the line above in your **`~/.bashrc`** file.

Now add the lines below to your ~/**`.bash_profile`** to assure it calls the ~/**`.bashrc`** file. It is not necessary to have this in your `~/.bash_profile` more than once:

```
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi
```

Exit your shell and start a new session to test that your configuration is able to correctly locate the executables by issuing these `which` commands in your shell:

```
$ which qbsolv
```

```
$ which dw-exec
```

If you see an "unknown command" or empty response to either of these commands, check your ~/**`.bashrc`** file and make sure that the location specified for the qOp executables is correct before proceeding.

After completing these configuration steps, you should be able to start a new shell and invoke `qbsolv` as follows:

```
$ qbsolv -V
```

You should see output like the following:

```
Version open source 2.5
Compiled: Sep 18 2017,13:36:05
```

that indicates qbsolv has executed properly.

Also try

```
$ dw
```

You should see output that looks like this:

```
USAGE: dw {get|set|cd|pwd|embed|bind|exec|val|add|trans|ls|mv|cp|touch|rm|...
```

If you get an error such as '`libdwave_sapi.dylib could not be found`' your `DYLD_LIBRARY_PATH` environment variable may not be set properly.

## C. Make changes so non-Terminal-started applications will run properly

In addition to making these changes to your ~/**.profile**, you'll need to apply the following configuration changes so that applications (such as Microsoft Excel) that are not started from Terminal will have the correct environment. In your Terminal window, issue these commands:

```
$ launchctl setenv DWAVE_HOME $DWAVE_HOME
$ launchctl setenv DYLD_LIBRARY_PATH $DYLD_LIBRARY_PATH     # see note below
```

You can check that these two commands have been successful by issuing the following commands:

```
$ launchctl getenv DWAVE_HOME
$ launchctl getenv DYLD_LIBRARY_PATH
```

Both the above commands should return a string equal to the value set for the DWAVE_HOME environment variable.

Note: the two commands in red should be omitted on El Capitan or later OS X versions.

## D. Change the ~/.dwrc file to connect to your D-Wave system

In the `$DWAVE_HOME` directory, you will see a file named `.dwrc`. This contains configuration information that enables you to quickly point any of the qOp tools to your D-Wave hardware system or to a D-Wave software simulator included in the D-Wave software. Copy that file to your home directory, for example via

```
$ cd
$ cp $DWAVE_HOME/.dwrc .
```

The released `.dwrc` file contains a single entry consisting of

```
laptop|local
```

where `laptop` is the name you would use to refer to it and `local` is a distinguished name indicating to use the simulator running on the system where qOp is installed. Running in the simulator is the easiest way to get started, as it eliminates some possible sources of failure.

At some point you will want to run on your D-Wave hardware system, so you will need to add an entry pointing to that. The format of such an entry is "`name|sapiURL,token`" or "`name|sapiURL,token,proxyURL`" if your D-Wave system is accessed via a proxy server. You'll probably want to contact the system administrator for your D-Wave system to get the SAPI URL and optionally proxy URL. Your token needs to be created via the Qubist interface. You will also want to learn the name(s) of the solvers on that system to which you have access. An example entry in `~/.dwrc` might be

```
mydwave|https://qubist.myorg.com/sapi,87e154d397123c40db123c
```

Once you've added this to your `~/.dwrc` file, you can direct execution of qOp programs to the simulator via

```
$ dw set connection laptop
$ dw set solver c4-sw_sample
$ qbsolv -i program.qubo
```

or to your D-Wave hardware system named `mydwave` via

```
$ dw set connection mydwave
$ dw set solver <mysolvername>
$ qbsolv -i program.qubo
```

At this point you are ready to create programs in any of the qOp tool input formats and execute them locally on the simulator or on a D-Wave system.

**Note**: The qOp 2.5 release includes pre-built workspaces for virtual-full-yield (VFY) solvers and depends on a new manual process for creating a pre-built workspace for a given D-Wave system. If you run on a D-Wave system and not via VFY solvers, please contact your D-Wave system administrator for access to the workspace specific to your D-Wave system. Similarly, if there is an error in setting up these pre-built workspaces, qbsolv may put out a message "No pre-embedding found", in which case please contact your D-Wave system administrator.

## E. Install the Python wrapper for qbsolv

As of qOp 2.5, qbsolv has a Python wrapper (`dwave_qbsolv` ) that is callable as a Python function. The simplest way of installing `dwave_qbsolv` is the following, which can be executed independent of the current directory:

```
$ pip install dwave_qbsolv
```

Some platforms or releases may not have pre-built binaries, in which case the method above will not work and building from source is required, which can be done by the following:

```
$ cd $DWAVE_HOME/src/qbsolv/python
$ pip install -r requirements.txt
$ python setup.py install
```

Whichever method you use for installing `dwave_qbsolv`, to confirm it is properly installed, you can execute the `tryDwaveQbsolv.py` example from the `examples/qbsolv` directory.

## F. Run the provided example programs

In the `$DWAVE_HOME/examples` directory are several examples intended to be useful, and scripts that build or execute them. All of the scripts whose names end in `.bash` are intended to be executed with the bash shell.

Use the qOp-examples command (alternatively, **`dw examples`**) to compile and run all the examples. The examples illustrate use of qbsolv, dw, ToQ, qsage, and C/SAPI. Run them as follows:

```
$ qOp-examples
```

This command will copy the contents of $DWAVE_HOME/examples to a new directory within your current working directory. After copying the files, the command will compile the various examples as necessary. You should see a single line of output for each C program compiled by the shell script, *e.g.*:

```
**************** C: Compiling sapi_connectingToSolver ****************
```

Additionally, you should see a few lines for each QUBO embedded into the simulator:

```
**************** dw: Embedding 1of3 ***************
```

After preparing all examples to run, the command will execute each example. Standard output from each of these jobs is captured in a file whose name matches the stem of the input file and whose suffix is .out.

The `$DWAVE_HOME/doc` directory contains "man" pages for the ToQ, Quantum Apprentice, `dw`, `qsage`, `qbsolv`, and `ToQ` tools, as well as the `.qubo` and `.q` file formats. (The user guides for the C Solver API are also in that directory.)

## G. If needed, create pre-embedding files specific to your system

The `qbsolv` tool uses a pre-embedded file to save time when executing. If you anticipate running on solvers that are specific to the D-Wave system you use, you may need to create embedding(s) that are specific to the system you use. If you only use virtual-full-yield (VFY) solvers, you can skip this step. If, however, you use solvers that are exactly configured to the hardware of the system you use, you need to run the `makefull.sh` script that resides in the `bin` directory in the qOp release. Start with the file `bin/ReadMemakefull.txt` for instructions.

## H. Copy `toq` configuration files into all directories where `toq` programs will be executed

The simplest way to configure the qOp tools to a particular execution target, such as the software simulator or your specific D-Wave system, is via `dw set`, as described just above. If you use that approach, you can skip this section.

If you do not use the `dw set` approach, there are several files in your `$DWAVE_HOME/examples` directory that make it more convenient to run `toq`. If you only run `toq` in the `examples` directory, you need not do anything, but if you want to run in some other directory (say `$HOME/mytoq`), there is one file you must copy, and several others that may make it easier to run `toq` programs. The file you will need is the `toq` startup file, `.toqrc`, which sets the `toq` configuration by default to run on the simulator. To run `toq` in `$HOME/mytoq`, enter the following commands:

```
$ cd $DWAVE_HOME/examples
$ cp -p .toqrc $HOME/mytoq
$ cd $HOME/mytoq
$ ./toq -r -i mypgm.toq
```

This final command will execute `toq` on your program `mypgm.toq`.

You can place the other configuration files into `$HOME/mytoq` by executing the following commands (note that these files provide shortcuts, but are not required to execute `toq`).

```
$ cd $DWAVE_HOME/examples
$ cp -p toqREADME confSim confHdw15 confHdw22 toqSim toqHdw  toqHdw15
        toqHdw22  toqSim2  $HOME/mytoq
```

(Note that this last command is all on one line.) Review the file `toqREADME` for background on how to use these configuration files.

# Appendix A: Contents

qOp contains the following components:

1. Quantum Apprentice: an Excel spreadsheet, which allows the user to manipulate, visualize and execute problems on the simulator and also quantum hardware.
2. ToQ: compiles and executes problems written in a prototype language-input format which defines a constraint satisfaction problem.
3. qsage: optimizes a user-provided C-language objective function.
4. qbsolv: executes quadratic unconstrained binary optimization (QUBO) programs, even if they happen to be bigger than will fit in the D-Wave hardware or simulator being used.
5. dw: provides shell access to many common API functions such as creating a connection, obtaining a solver, initializing and executing a QMI, and examining samples.
6. Solver API (SAPI) C library: invokes the low-level D-Wave interface routines. It contains a standard header file and accompanying library file. You can use it to compile and link C programs that directly use the low-level API. Example code is provided which shows how to do this. A separate document describes this API in detail.

Before installing qOp, you may need to install and/or configure certain software packages upon which qOp depends.

# Appendix B: Prerequisites for C/C++-language programs

## A. Installing XCode

You will need to install and configure the Xcode package. This will allow you to compile and link programs using the D-Wave C language API.

### Xcode

You can read about Xcode here:

https://developer.apple.com/xcode/

Xcode is a complete developer's toolkit for the OS X environment. To use the D-Wave qOp package you will only need the Xcode Command Line Tools. Instructions for obtaining this package can be found here:

http://osxdaily.com/2014/02/12/install-command-line-tools-mac-os-x/

Briefly, you will start your Terminal and issue this command:

```
$ xcode-select –install
```

Click "Install" in the pop-up dialog box, then agree to the terms of service. The download is around 130 Mbytes.

## B. Confirming the C and C++ compilers work

Open an OS X shell window. At the prompt, type

```
$ which gcc
```

```
$ which g++
```

If the output from either of these commands is “unknown command” or an empty response, then you will need to install the compilers.

Let’s also check the version of the gcc and g++ commands as follows:

```
$ gcc --version
```

```
$ g++ --version
```

The version results on a sample OS X system look like this:

```
Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr --
with-gxx-include-
dir=/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Deve
loper/SDKs/MacOSX10.11.sdk/usr/include/c++/4.2.1
Apple LLVM version 7.0.2 (clang-700.1.81)
Target: x86_64-apple-darwin15.3.0
Thread model: posix
```