Building BOINC Client and Manager on Macintosh

Written by Charlie Fenton
Last updated 10/11/2025

This document applies to BOINC version 8.3.0 and later. It has instructions for building the BOINC Client, Manager, installer and associated software for the Macintosh. Information for building science project applications to run under BOINC on Macintosh can be found here and here.

Note: the information in this document changes from time to time for different versions of BOINC. For any version of BOINC source files, the corresponding version of this document can be found in the source tree at:

`boinc/mac_build/HowToBuildBOINC_XCode.rtf`

Contents of this document:

Note: `setupForBoinc.sh` (described later in this document) runs `buildWxMac.sh` to build the wxWidgets library used by BOINC Manager. If you built the wxWidgets library with an earlier version of `buildWxMac.sh` and the BOINC Manager build fails with linker (`ld`) errors, then you may need rebuild wxWidgets with the `buildWxMac.sh` included in the newer source tree.

Important requirements for building BOINC software for the Mac

As of version 6.13.0, BOINC does not support Macintosh PowerPC processors. As of 7.16.14, BOINC is built as Universal2 Binaries which can run on both 64-bit Intel and Apple Silicon (arm64) hardware.

You need to take certain steps to ensure that you use only APIs that are available in all the OS versions BOINC supports for each architecture. The best way to accomplish this is to use a single development system running versions of MacOS and Xcode which are / were current at the time your target version of BOINC was released, and cross-compile for the various architectures. The remainder of this document describes that process.

The above requirements apply not only to BOINC itself, but also to the wxWidgets, c-ares, cURL, openSSL, freetype and ftgl libraries, as well as all project applications.

Be sure to follow the directions in this document to ensure that these requirements are met.

Installing and setting up Xcode

Apple provides the tools necessary to build BOINC on any Mac running MacOS 15 or later. You get these tools, including the GCC or Apple LLVM compiler and system library header files, by installing the Xcode Tools package. Note that the current version of BOINC requires Xcode 16 or later.

You can download Xcode from Apple's App Store (it is large: over 4 GB). If you are a member of Apple's Mac Developer Program, you can also download it from Apple's web site: `http://developer.apple.com`.

If Xcode is obtained from the Apple Store then it will be installed automatically into the Applications folder. Double-click on the installed Xcode icon to run Xcode. Xcode will display a dialog allowing you to finish the installation by installing its associated command-line tools; you must do this before running BOINC's build scripts. (Some versions of Xcode may not display this dialog until you open a file with Xcode.)

Building BOINC Manager with embedded Core Client

Source files are now archived using git. For details, see:
`https://github.com/BOINC/boinc/wiki/SourceCodeGit`

BOINC depends on six third-party libraries: wxWidgets, c-ares, curl, openssl, freetype and ftgl. Different versions of BOINC require different versions of these libraries; any one version of BOINC will build with only one specific version of each library. For BOINC versions prior to 8.3.0, see `mac_build/HowToBuildBOINC.rtf` in the source tree for that version of BOINC for the library versions needed and how to download them.

Starting with BOINC 7.24.0, the `SetupForBOINC.sh` script (described below) in the source tree for any version of BOINC automatically downloads and builds the appropriate six third-party libraries for that version of BOINC.

More information about each third-party library can be found on its home page:
wxWidgets- (needed only if you are building the BOINC Manager): `https://wxwidgets.org`
curl: `https://curl.se`

c-ares (used by curl): https://c-ares.org
openssl: http://www.openssl.org/
freetype (needed only if you are building the BOINC default screensaver or a project screensaver): http://www.freetype.org/
ftgl (needed only if you are building the BOINC default screensaver or a project screensaver): http://sourceforge.net/projects/ftgl

IMPORTANT: The third-party libraries may not work properly with BOINC if you build them directly with either the Xcode projects which come with wxWidgets or the AutoMake scripts supplied with all six, so be sure to use our special scripts to build these packages.

[1] Make sure you are logged into the Mac using an account with administrator privileges. Create a parent directory within which to work. In this description; we will call it BOINC_dev, but you can name it anything you wish.

[2] If you have not yet done so, install Xcode and launch it once to accept the license agreement and complete the installation.

[3] Get the BOINC source tree from the repository, and put it in the BOINC_dev folder. To do this, type the following in Terminal (if you have problems, you may need to enter `sudo` and a space before the `git` command):

```
cd {path}/BOINC_dev/
git clone https://github.com/BOINC/boinc.git boinc
```

(You may change the name of the `boinc` directory to anything you wish.)

The command above retrieves the source code from the HEAD / MASTER (TRUNK) or development branch of the git repository. You can also get or switch to the source code for any released version of BOINC. For more information on getting the BOINC source code, see:
https://github.com/BOINC/boinc/wiki/SourceCodeGit

[4] Run the script to download and build the c-ares, curl, openssl, wxWidgets, freetype and ftgl libraries as follows:

```
cd {path}/BOINC_dev/boinc/mac_build/
source setupForBoinc.sh -clean
```

The above commands will download these libraries to the BOINC_dev folder (if they haven't previously been downloaded there.) If you don't wish to force a full rebuild of everything, omit the -clean argument.

Note 1: Be sure to run the script using the `source` command. Do not double-click on the scripts or use the `sh` command to run them.
Note 2: This script tries to build all six third-party libraries: wxWidgets, c-ares, curl, openssl, freetype and ftgl. When the script finishes, it will display a warning about any libraries it was unable to build. To make it easier to find the error messages, clear the Terminal display and run the script again without `-clean`.
Note 3: `setupForBoinc.sh` runs `buildWxMac.sh` to build the wxWidgets library used by BOINC Manager. If you previously built the wxWidgets library with an earlier version of `buildWxMac.sh` and the BOINC Manager build fails with linker (`ld`) errors for symbols starting with "wx", you may need to rebuild it with the `buildWxMac.sh` included in the newer source tree.
Note 4: The *{path}* must not contain any space characters.
Hint: You don't need to type the path to a file or folder into Terminal; just drag the file or folder icon from a Finder window onto the Terminal window.

[5] Build BOINC as follows:

BOINC itself is built using the **boinc.xcodeproj** file. You can either build directly in Xcode (more information below) or run the **BuildMacBOINC.sh** script:

```
cd {path}/BOINC_dev/boinc/mac_build/
source BuildMacBOINC.sh
```

The complete syntax for this script is

```
source BuildMacBOINC.sh [-dev] [-noclean] [-libstdc++] [-all] [-lib] [-client] [-zipapps] [-uc2] [-vboxwrapper] [-target targetName] [-setting name value] [-help]
```

The options for BuildMacBOINC.sh are:

-dev — Build the development (debug) build. Default is deployment (release) build.

-noclean — Don't do a "clean" of each target before building. Default is to clean all first.

-libstdc++ — Build using libstdc++ instead of libc++ (see Note 4 below.)

The following 6 arguments determine which targets to build:

-all — Build all targets (i.e. target "Build_All" -- this is the default) **except** boinc_zip_test, testzlibconflict, UpperCase2 targets (UC2-x86_64, UC2Gfx-x86_64 and slide_show-x86_64) and VBoxWrapper.

-lib — Build the six libraries: libboinc_api.a, libboinc_graphics_api.a, libboinc.a, libboinc_opencl.a, libboinc_zip.a, jpeglib.a.

-client Build two targets: boinc client and command-line utility boinccmd
(also builds libboinc.a, since boinc_cmd requires it.)

You should not need to build these three targets:

-zipapps Build two zip samples: boinc_zip_test and testzlibconflict.

-uc2 Build the UpperCase2 samples: UC2-x86_64, UC2Gfx-x86_64 and slide_show-x86_64.

-vboxwrapper Build the VBoxWrapper target (used only for project applications.)

-docker_wrapper Build the Docker Wrapper target (used only for project applications.)

Both -lib and -client may be specified to build seven targets (no BOINC Manager or screensaver.)

The following arguments are used mainly for building the daily test builds and are not likely to be useful to you:

-target targetName Build *only* the one target specified by targetName

-setting name value Override setting 'name' to have the value 'value' (usually used along with -target.)
You can pass multiple -setting arguments.)

Note 1: boinc.xcodeproj in the BOINC_dev/boinc/mac_build/ directory builds BOINC. It can be used either with the `BuildMacBOINC.sh` script or as a stand-alone project. The *Development* build configuration is used for debugging. The *Deployment* build configuration builds a universal binary and is suitable for release builds. If there are any other build configurations, they should not be used as they are obsolete.

Note 2: To perform a release build under Xcode 16 or later when not using the `BuildMacBOINC.sh` script, select "Build for Profiling" from Xcode's Product menu. To save disk space, do **not** select "Archive."

Note 3: Using the `BuildMacBOINC.sh` script is generally easier than building directly in Xcode. The script will place the built products in the directory `BOINC_dev/boinc/mac_build/build/Deployment/` or `BOINC_dev/boinc/mac_build/build/Development/` where they are easy to find.

Building directly in Xcode places the built products in a somewhat obscure location. To determine this location, control-click on *Products* in Xcode's Project Navigator and select "Show in Finder", or use the method in **Note 5** below. Building the jpeglib.a and 5 libboinc*.a libraries will copy them to the directory `BOINC_dev/boinc/mac_build/build/Deployment/` or `BOINC_dev/boinc/mac_build/build/Development/.`
cd
Note 4: As of version 7.15.0, BOINC is always built using libc++. Project applications built for libstdc ++ with newer versions of Xcode will not link properly with BOINC libraries built for libc++.

Hint: You can install multiple versions of Xcode on the same Mac, either by putting them in different directories or by renaming Xcode.app of different versions. You can then specify which version the `BuildMacBOINC.sh` script should use by setting the `DEVELOPER_DIR` environment variable using the `env` command. For example, if you have installed Xcode 16.2 in the Applications directory and renamed `Xcode.app` to `Xcode_16_2.app`, you can invoke the script with:

```
env DEVELOPER_DIR=/Applications/Xcode_16_2.app/Contents/Developer sh BuildMacBOINC.sh
```

Note 5: When building the client or BOINC Manager, the BOINC Xcode project has built-in scripts which create a text file with the path to the built products at either `BOINC_dev/boinc/mac_build/Build_Deployment_Dir` or `BOINC_dev/boinc/mac_build/Build_Development_Dir`. These files are used by the `release_boinc.sh` script, but you can also use them to access the built products directly as follows; open the file with TextEdit and copy the path, then enter command-shift-G in the Finder and paste the path into the Finder's dialog.

Note 6: Important: As of Xcode 15.3, if you build directly with Xcode it will by default build only for the native architecture. Xcode 15.3 added a new “Override Architectures” build option, which controls the set of architectures that will be built for all targets in the workspace. The default is “Match Run Destination”, which will not build a universal binary (x86_64 and arm64) and will cause a link error for the *detect_rosetta_cpu* target on Apple Silicon Macs. To fix this, edit the schemes for each target you wish to build, and in the scheme's `Build` section change the `Override Architectures` option to `Use Target Settings`. If you are building the `Build_All` target, it is sufficient to do this only for that target. This step is not necessary if you build using the **BuildMacBOINC.sh** script or if using an older version of Xcode to build an earlier version of BOINC.

Building BOINC Manager Installer

In order to execute BOINC Manager, you must install it using the BOINC Manager Installer (unless running the development build under the Xcode debugger.) Otherwise, you will encounter an error prompting for proper installation.

To build the Installer for the BOINC Manager, you must be logged in as an administrator. If you are building BOINC version number x.y.z, type the following in Terminal, then enter your administrator password when prompted by the script:

```
cd {path}/BOINC_dev/boinc/
source mac_installer/release_boinc.sh x y z
```

Substitute the 3 parts of the BOINC version number for x y and z in the above. For example, to build the installer for BOINC version 8.3.0, the command would be

```
source mac_installer/release_boinc.sh 8 3 0
```

This will create a directory "BOINC_Installer/New_Release_8_3_0" in the BOINC_dev directory, and the installer will be located in the directory: *"{path}*/BOINC_dev/BOINC_Installer/New_Release_8_3_0/boinc_8.3.0_macOSX_universal/"

The installer script uses the deployment (release) build of BOINC; it won't work with a development (debug) build.
You can find the current version number in the file *{path}*/BOINC_dev/boinc/version.h

Code Signing the BOINC Manager Installer and Uninstaller

Mac OS 10.8 introduced a security feature called Gatekeeper, whose default settings won't allow a user to run applications or installers downloaded from the Internet unless they are signed by a registered Apple Developer. The `release_boinc.sh` script looks for a file `~/BOINCCodeSignIdentities.txt` containing the name of a valid code signing identity stored in the user's Keychain (where ~ represents the path to your home directory, typically something like "/Users/JohnSmith".) If this is found, the script will automatically sign the BOINC installer and BOINC uninstaller using that identity. For example, if your user name is John Smith, the first line of `~/BOINCCodeSignIdentities.txt` would be something like the following:

```
Developer ID Application: John Smith
```

If you wish to also code sign the installer package, add a second line to `~/BOINCCodeSignIdentities.txt` with the installer code signing identity. This would be something like the following:

```
Developer ID Installer: John Smith
```

If there is no `~/BOINCCodeSignIdentities.txt` file, then the script will not sign the installer or uninstaller. Code signing is not necessary if you won't be transferring the built software over the Internet. For more information on code signing identities enter "`man codesign`" in Terminal and see Apple's `Code Signing Guide` and `Tech Note 2206`.

Important: As of MacOS 10.15 Catalina, the OS does not allow the user to run software downloaded from the Internet unless it has been "notarized" by Apple. You can notarize BOINC only if you have code signed everything, including the installer package. Notarizing is not necessary if you won't be transferring the built software over the Internet. If you are building BOINC version number x.y.z, type the following in Terminal to notarize the installer and uninstaller:

```
cd {path}/BOINC_dev/boinc/
source ./mac_installer/notarize_boinc.sh x y z
```

Substitute the 3 parts of the BOINC version number for x y and z in the above using the same values you used when building the installer.

You must have done the following **before running this script:**
- Created an app-specific password by following the instructions on "Using app-specific passwords" here.
  NOTE: You cannot use your normal Apple ID password.
- Created a profile named "notarycredentials" in your keychain using the "notarytool store-credentials" command. (For details, enter "man notarytool" in Terminal and look at the description for the `store-credentials` subcommand.)
- Run the release_boinc.sh script as described above.

For more information, type the following in Terminal:

```
xcrun notarytool --help
man stapler
```

Debugging and BOINC security

Version 5.5.4 of BOINC Manager for the Macintosh introduced new, stricter security measures. For details, please see the file `BOINC_dev/boinc/mac_installer/Readme.rtf` and `https://boinc.berkeley.edu/sandbox_design.php` and `https://github.com/BOINC/boinc/wiki/SandboxUser`

The LLDB debugger can't attach to applications which are running as a different user or group so it ignores the S_ISUID and S_ISGID permission bits when launching an application. To work around this, the BOINC *Development* build does not use the special boinc_master or boinc_project users or groups, and so can be run under the debugger from Xcode. This also streamlines the development cycle by avoiding the need to run the installer for every change. (To generate the development build under Xcode, choose "Build" from the product menu, or enter command-B on the keyboard.)

To restore the standard ownerships and permissions, run the BOINC installer or run the BOINC_dev/boinc/mac_build/Mac_SA_Secure.sh shell script in Terminal (the comments in this script have instructions for running it.)

For information on interpreting crash dumps and backtraces, see `Mac Backtrace`.

Debugging into wxWidgets

The BOINC Xcode project links the BOINC Manager with the non-debugging (Deployment) build of wxWidgets for the Deployment build configuration of the Manager, and with the debugging (Development) build of wxWidgets for the Development build configuration of the Manager. You should use the Development build of the Manager and wxWidgets for debugging; this allows you to step into internal

wxWidgets code. With the Development build, you can even put breakpoints in wxWidgets; this is most easily done after stepping into a function in wxWidgets source file containing the code where you wish to put the breakpoint.