Build Instructions For Mac

Patrick Heyer edited this page on Apr 2 · 10 revisions

NOTE: Since March 2023, `obs-studio` uses an updated build system on macOS that automated most steps required in old build systems. Build instructions for the legacy build system are retained in this document, albeit in a simplified form reflecting the suggested way macOS builds should be approached.

Prerequisites

- macOS 12.0 (for Xcode versions *up to and including 14.2*)
- macOS 13.0 (for Xcode versions *after and including 14.3*)
- Xcode 14.2 or newer
- CMake 3.24 or newer
- CCache 4.8 or newer (*Optional*)

Configuring Build Project

1. Clone the repository including **submodules**:

```
git clone --recursive https://github.com/obsproject/obs-studio.git
```

2. Check available CMake presets `cmake --list-presets`
3. Select preset for desired build architecture (`macos-arm64` or `macos-x86_64`): `cmake --preset macos-arm64`

Build obs-studio

1. Open the Xcode project file in the generated build directory (`build_arm64` or `build_x86_64`)
2. Select the `obs-studio` scheme from the scheme browser at the top of the Xcode window
3. Press `<CMD>+<B>` to build the project (or `<CMD>+<R>` to build and run it).

Alternatively the project can also be built on the command line:

1. Change into the generated build directory (`build_arm64` or `build_x86_64`)
2. Run `xcodebuild -configuration <Debug|Release|RelWithDebInfo|MinSizeRel> -scheme obs-studio -parallelizeTargets -destination "generic/platform=macOS,name=Any Mac"`
3. Run the app via `open UI/<Debug|Release|RelWithDebInfo|MinSizeRel>/OBS.app`

HINT: You can install the tool `xcbeautify` to make the `xcodebuild` output more readable - the suggested way to run the build command changes slightly in that case: `set -o pipefail && <xcodebuild command> 2>&1 | xcbeautify`

Installation

CMake creates an `install` scheme that will automatically install OBS into the `Applications` directory. Choose this scheme from within Xcode (or specify via the command line) and build the project.

Archive and Export via Xcode

To generate optimised and fully codesigned application bundles, use the archive and export functions provided by Xcode:

1. Open the Xcode project

2. Select "Archive" from the "Product" menu item (Xcode will build the project in "Release" configuration, as is canonical on macOS)
3. Xcode will automatically open the "Organizer" window after successfully creating an archive
4. Select "Distribute App" to start an export of the archive
5. Select "Copy App" to just copy OBS to a desired location
6. Selecting "Developer ID" allows one to "Export" or "Upload" the application
   - "Export" checks if the application is fully compliant with Apple's codesigning requirements (you will be prompted to specify a provisioning profile for specific targets if necessary)
   - "Upload" checks if the application is fully compliant and also uploads it to Apple's notarization service
   - Both options require the project to be configured with a correct developer certificate, team identifier and provisioning profile (see below)

Custom Build Options

Custom build options can be provided to CMake after specifying the CMake preset:

- Either specify them directly as cache variables, e.g. `cmake --preset macos-arm64 -DENABLE-BROWSER:BOOL=OFF`
- OR run the CLI UI with a preset to change options directly: `ccmake --preset macos-arm64`

Codesigning

Starting with macOS 11, it is best practice to *always* codesign all binaries on macOS. For local execution an ad-hoc signature is sufficient, which Xcode will handle automatically ("Sign to Run Locally").

Because macOS stores application permissions (e.g., webcam access, microphone access, screen recording) based on code signatures, this has the side-effect of permissions not being applied to updated binaries (the underlying hashes will not match the entry in macOS' database).

The only way to fix this is to compile the application with a valid Apple developer certificate, as the code signature will not change between builds of the application and permissions will "stick".

Configure Codesigning

Because `obs-studio` is made up of multiple independent parts, signing is most easily configured via CMake: Specify a developer team ID via `-DOBS_CODESIGN_TEAM` (you can find your team identifier in the Apple developer portal). This will enable automatic codesigning in Xcode for all targets built by the project.

When building (and packaging) on CI, a developer identity need to be provided instead via `-DOBS_CODESIGN_IDENTITY`. An identity commonly takes a form of a string like `Developer ID Application: <YOUR_NAME> (<YOUR_TEAM_ID>)`. This needs to match the name of the developer certificate that is installed on the machine.

To specify a provisioning profile (needed in future versions of `obs-studio` with certain macOS features enabled), pass either the *name* or *UUID* of the provisioning profile to CMake via `-DOBS_PROVISIONING_PROFILE`. If the provisioning profile is correctly installed in the system (usually by importing it via Xcode or copying it to `~/Library/MobileDevice/Provisioning\ Profiles`) Xcode will automatically pick it up. It will also be used for archiving and exporting.

Legacy Build Instructions

NOTE: The following instructions only apply to the legacy build system superseded by the instructions above. They are retained here for the time being to allow building older versions of `obs-studio` for now-unsupported versions of macOS.

Custom macOS builds allow full customization of the desired build configuration but also require manual setup and preparation. Available CMake configuration variables can be found in the CMake build system documentation.

Prerequisites

- macOS 10.15 or newer
- Command Line Tools (CLT) for Xcode:

- Install by running `xcode-select --install` in Terminal, **or**
- Download from https://developer.apple.com/downloads, **or**
- Install Xcode
- CMake
- Ninja
- *Optional:* CCache to improve compilation speeds on consecutive builds
- Pre-Built OBS dependencies (includes `FFmpeg`, `x264`, `mbedTLS`, `Qt` and more)
  - Download the latest version for your desired architecture from https://github.com/obsproject/obs-deps/releases
  - Unpack the contents of the archive into `~/development/obs-build-dependencies/obs-deps`
- For browser source and browser panel support, the pre-built CEF framework is needed:
  - Chromium Embedded Framework (CEF) Intel x86_64
  - Chromium Embedded Framework (CEF) Apple M1
  - Preparation of CEF for use in OBS requires some fixups and compilation of the static wrapper library (see below)
- Alternatively download and setup the following dependencies manually:
  - FFmpeg
  - x264
  - freetype
  - mbedtls
  - swig
  - Qt 6.3

Build procedure

1. Get the source code

1. Open a Terminal window, create and switch to a directory you want to have OBS checked out
2. Clone the repository including **submodules**: `git clone --recursive https://github.com/obsproject/obs-studio.git`

(If you do not know what submodules are, or you are not using Git from the command line, **PLEASE make sure to fetch the submodules too**.)

2. Get the dependencies

- To download and set up most preconditions mentioned above, you can also run the script `CI/macos/01_install_dependencies.sh` from the checkout directory (run it with the `--help` switch to see all available options).

NOTE: The directory where the script will download and setup the dependencies in cannot be changed.

3. Set up the build project

1. Run CMake to generate a build environment

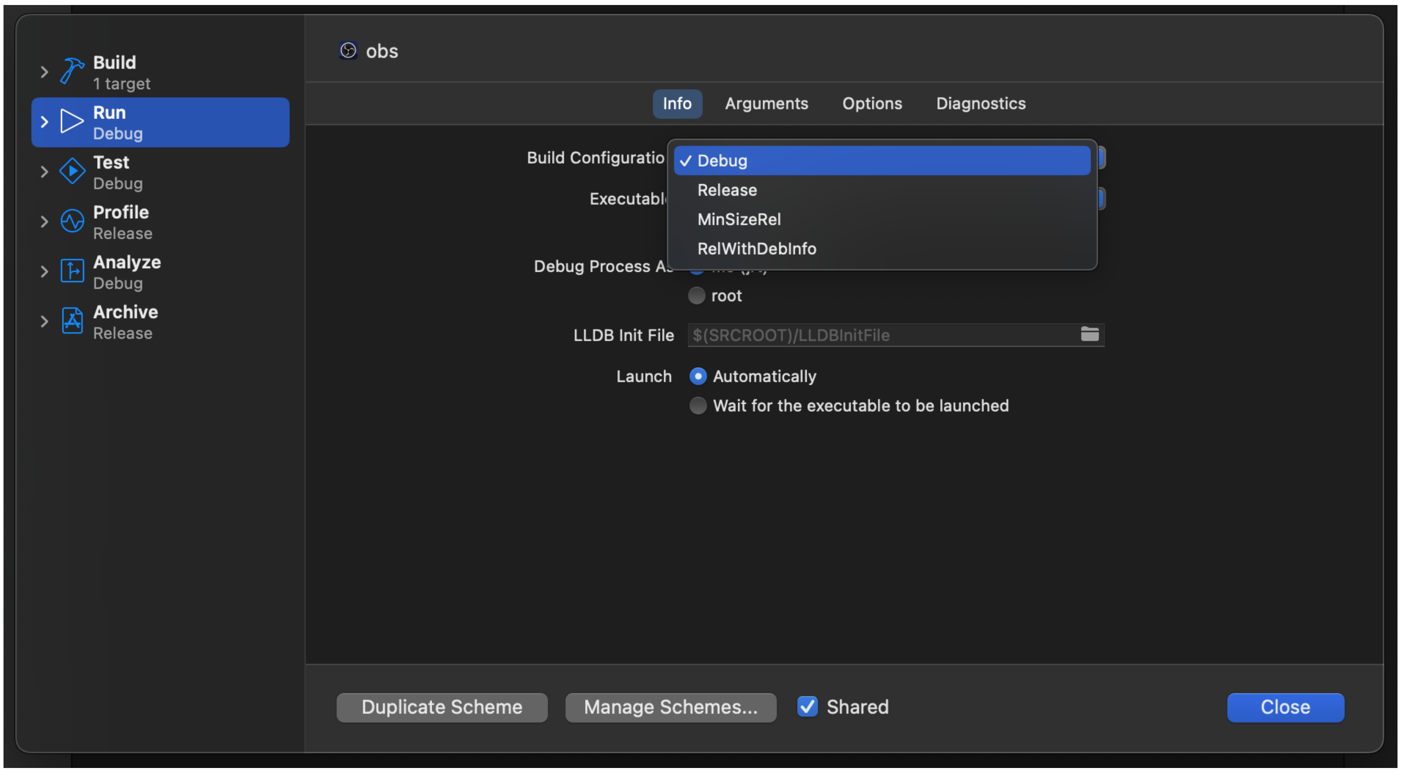
```
cmake -S . -B build -G Xcode \
    -DCEF_ROOT_DIR="~/development/obs-build-dependencies/cef_binary_5060_macos_x86_64" \
    -DCMAKE_PREFIX_PATH="~/development/obs-build-dependencies/obs-deps" \
    -DCMAKE_OSX_DEPLOYMENT_TARGET=11.0 \
    -DCMAKE_OSX_ARCHITECTURES="x86_64"
```

Optional Settings:

- Specify a codesigning identity using `-DOBS_BUNDLE_CODESIGN_IDENTITY="[YOUR_IDENTITY]"` or a team using `-DOBS_BUNDLE_CODESIGN_TEAM="[YOUR_TEAM_ID]"`
- Specify the output directory for creating application bundles using `-DCMAKE_INSTALL_PREFIX=[YOUR INSTALL DESTINATION]`.
- Build using Ninja or UNIX Makefiles with `-G Ninja` or `-G 'Unix Makefiles'`
  - Specify the build configuration e.g. using `-DCMAKE_BUILD_TYPE=RelWithDebInfo`

4. Open the Xcode project

1. Open `obs-studio.xcodeproj` from the build directory (or any other directory specified via the `-B` switch above)
2. Build OBS Studio using `obs` schema, recognisable by the OBS Studio app icon.
3. To debug OBS Studio, make sure the Run scheme's Build Configuration is set to Debug. (This is the default.)



Build Configuratio ✓ Debug
Release
MinSizeRel
RelWithDebInfo
Executabl
Debug Process A
root
LLDB Init File $(SRCROOT)/LLDBInitFile
Launch ● Automatically
Wait for the executable to be launched

5. Install OBS.app bundle

Installation will use the directory specified via `-DCMAKE_INSTALL_PREFIX` or can be customised with the `--prefix` switch for Ninja-based builds.

- If using Xcode:
  - Open the generated Xcode project
  - Install OBS using the `install` schema
- If using the command line (Xcode, Ninja, or GNU Makefiles):
  - Build OBS by running `cmake --build build`
  - Install OBS by running `cmake --install build`

obsproject.com

▸ **Pages** 69

Home

Install instructions

- Windows
- macOS
- Linux
- FreeBSD

Build instructions

- Windows

- macOS
- Linux
- FreeBSD

Development & Scripting

- Getting Started with OBS Studio Development
- Getting Started With OBS Scripting
    - Scripting Tutorial: Source Shake
    - Scripting Tutorial: Halftone Filter
- Service Submission Guidelines

Clone this wiki locally

```
https://github.com/obsproject/obs-studio.wiki.git
```