Build Instructions For Windows

JonasCz edited this page on Mar 1 · 20 revisions

Option A: Automatic Windows builds

Automatic Windows builds allow building OBS with minimal input and setup. Necessary dependencies are installed automatically, build flags use a sane default, and the generated OBS build uses the application's full feature set.

Prerequisites

- Windows PowerShell (v5+ and more recent)
- Visual Studio 2022 (at least Community Edition)
  - Windows 10 SDK (minimum 10.0.20348.0)
  - C++ ATL for latest v143 build tools (x86 & x64)
- Git for Windows

Note that the automatic build scripts can use Chocolatey to automatically install additional build dependencies (CMake and 7-Zip), by passing the `-Choco` switch.

Build procedure

- Clone the repository including **submodules**:

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

- To do a **fully automated** build, open a PowerShell window, switch to the checkout directory then run one of the following commands:

```
# Download and set up dependencies, then build OBS for local host
# architecture with common feature set
CI/build-windows.ps1

# Check for dependencies installable via Chocolatey
CI/build-windows.ps1 -Choco

# Skip download and setup of dependencies
CI/build-windows.ps1 -SkipDependencyChecks

# Build 32-bit only
CI/build-windows.ps1 -BuildArch '32-bit'

# Build both architectures
CI/build-windows.ps1 -CombinedArchs

# Create a zip archive with OBS and all required libraries
CI/build-windows.ps1 -Package

# Create a debug build
CI/build-windows.ps1 -BuildConfiguration Debug

# Use `my_build_dir` prefix as build directory
CI/build-windows.ps1 -BuildDirectory my_build_dir

# Build and package a combined 64-bit and 32-bit of OBS with Release configuration,
```

```
# using more verbose output and skipping dependency checks
CI/build-windows.ps1 -SkipDependencyChecks -CombinedArchs -BuildConfiguration Release -Verbose

# Show all available options
CI/build-windows.ps1 -Help
```

Option B: Custom Windows builds

Custom Windows 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

- Visual Studio 2022 (recommended)
  - Windows 10 SDK (minimum 10.0.20348.0). Latest SDK
- Development packages of `FFmpeg` , `x264` , `cURL` , and `mbedTLS`
  - Pre-built Windows dependencies for Visual Studio 2022 can be found in the obs-deps repo releases
- Qt 6
  - You can download our build of Qt 6.3.1 from the obs-deps repo releases.

    OR
  - You can install the official Qt 6 distribution from the Qt website. Grab the MSVC package for your version of Visual Studio.
  - OBS officially builds with Qt 6.3.1, though you may be able to build with other versions of Qt.
- CEF Wrapper (x64, x86)
- Windows version of CMake (3.20 or higher, latest preferred)
- Windows version of Git (Git binaries must exist in path)

Build procedure

1. Get the source code

- 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

- Download and set up most preconditions mentioned above, you can also run the script `CI/windows/01_install_dependencies.ps1` (run it with the `-Help` switch to see all available options).

NOTE: You cannot change the directory where the script will download and setup the dependencies in.

3. Set up the build project

1. Run cmake-gui, and set the following fields:
   - In "where is the source code", enter in the repository directory (example: `D:/obs`).
   - In "where to build the binaries", enter the repository directory path with the 'build' subdirectory (example: `D:/obs/build`). If this directory does not exist, it will be created by CMake.
2. Set required CMake variables either as Windows environment variables (allows usage across multiple projects) or directly as cache variables. Check the CMake build system documentation for a full list and description of these variables:
   - `CMAKE_PREFIX_PATH` - **REQUIRED**

     Example: `D:/obs-build-dependencies/windows-deps-2022-08-02-x64`

   - `DepsPath` (`DepsPath32` and `DepsPath64` as architecture-specific variants) - **LEGACY**
   - `QTDIR` (`QTDIR32` and `QTDIR64` as architecture-specific variants) - **LEGACY**
   - `CEF_ROOT_DIR` (when building with browser support)

     Example: `D:/obs-build-dependencies/cef_binary_5060_windows_x64`
   - `VIRTUALCAM_GUID` (when building with Virtual Camera support)
3. In cmake-gui, press `Configure` , and select the generator that corresponds with the desired installed Visual Studio version:
   - Visual Studio 17 2022, **or their 64bit equivalents** if you want to build the 64bit version of OBS
   - **NOTE**: If you need to change your dependencies from a build already configured, you will need to uncheck `COPIED_DEPENDENCIES` and run `Configure` again.
4. If you did not set up environment variables earlier you can now configure the variables named above in cmake-gui
5. In cmake-gui, press `Generate` to generate Visual Studio project files in the `build` subdirectory.

4. Open the Visual Studio project

1. Open `obs-studio.sln` from the subdirectory you specified under "where to build the binaries" (e.g. `D:/obs/build`) in Visual Studio (or click the `Open Project` button from within cmake-gui).
2. The project should now be ready to build and run. All required dependencies should be copied on compile and it should be a fully functional build environment. The build artifacts are installed into a subdirectory called `rundir/<CONFIG>` within your chose build directory (with `<CONFIG>` being `Debug` , `RelWithDebInfo` or any other build configuration that was successfully built).

5. Install the virtual camera

If you want to use the Virtual Camera created by this build, you will have to run its install script and also remove the Virtual Camera from a standard OBS installation first:

- To uninstall the OBS Virtual Camera
  1. Close any applications that were using the OBS Virtual Camera.
  2. In the OBS Studio installation directory, run `data\obs-plugins\win-dshow\virtualcam-uninstall.bat` as administrator.
- To install an OBS Virtual Camera:
  1. In the OBS Studio artifact directory (for Visual Studio builds, this is `<BUILD DIRECTORY>/rundir/<CONFIG>`), run `data\obs-plugins\win-dshow\virtualcam-install.bat` as administrator.

Don't forget to uninstall your build's virtual camera before cleaning/deleting your build files.

6. Integrating clang-format into Visual Studio

Use of `clang-format` is required for pull requests, and OBS targets the version shipped with Visual Studio 2022 17.2, `clang-format 13.0.1` . To configure any Visual Studio installation to use `clang-format 13.0.1` :

1. Download and install LLVM 13.0.1.
2. Run Visual Studio, select `Tools -> Options` from the menu.
   - Go to `Text Editor -> C/C++ -> Code Style -> Formatting -> General` .
   - Enable "Use custom clang-format.exe" and enter the file name, e.g. `C:\Program Files\LLVM\bin\clang-format.exe` .
   - The default keyboard shortcut for formatting a document (Edit.FormatDocument) is Ctrl+K, Ctrl+D.

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
```