Attention: Here be dragons

This is the `latest` (unstable) version of this documentation, which may document features not available in or compatible with released stable versions of Godot.

See the stable version of this documentation page instead.

Compiling for Windows

See also

This page describes how to compile Windows editor and export template binaries from source. If you're looking to export your project to Windows instead, read Exporting for Windows.

Requirements

For compiling under Windows, the following is required:

- A C++ compiler. Use one of the following:
  - Visual Studio Community , version 2019 or later. Visual Studio 2022 is recommended. **Make sure to enable C++ in the list of workflows to install.** If you've already installed Visual Studio without C++ support, run the installer again; it should present you a **Modify** button. Supports `x86_64` , `x86_32` , and `arm64` .
  - MinGW-w64 with GCC can be used as an alternative to Visual Studio. Be sure to install/configure it to use the `posix` thread model. **Important:** When using MinGW to compile the `master` branch, you need GCC 9 or later. Supports `x86_64` and `x86_32` only.
  - MinGW-LLVM with clang can be used as an alternative to Visual Studio and MinGW-w64. Supports `x86_64` , `x86_32` , and `arm64` .

- Python 3.6+ . **Make sure to enable the option to add Python to the `PATH` in the installer.**
- SCons 3.1.2+ build system. Using the latest release is recommended, especially for proper support of recent Visual Studio releases.

Note

If you have Scoop installed, you can easily install MinGW and other dependencies using the following command:

```
scoop install gcc python scons make mingw
```

Note

If you have MSYS2 installed, you can easily install MinGW and other dependencies using the following command:

```
pacman -S mingw-w64-x86_64-python3-pip mingw-w64-x86_64-gcc  \
    mingw-w64-i686-python3-pip mingw-w64-i686-gcc make
```

For each MSYS2 MinGW subsystem, you should then run *pip3 install scons* in its shell.

See also

To get the Godot source code for compiling, see Getting the source.

For a general overview of SCons usage for Godot, see Introduction to the buildsystem.

Setting up SCons

To install SCons, open the command prompt and run the following command:

```
python -m pip install scons
```

If you are prompted with the message `Defaulting to user installation because normal site-packages is not writeable`, you may have to run that command again using elevated permissions. Open a new command prompt as an Administrator then run the command again to ensure that SCons is available from the `PATH`.

To check whether you have installed Python and SCons correctly, you can type `python --version` and `scons --version` into a command prompt (`cmd.exe`).

If the commands above don't work, make sure to add Python to your `PATH` environment variable after installing it, then check again. You can do so by running the Python installer again and enabling the option to add Python to the `PATH`.

If SCons cannot detect your Visual Studio installation, it might be that your SCons version is too old. Update it to the latest version with `python -m pip install --upgrade scons`.

Downloading Godot's source

Refer to Getting the source for detailed instructions.

The tutorial will assume from now on that you placed the source code in `C:\godot`.

> **Warning**
>
> To prevent slowdowns caused by continuous virus scanning during compilation, add the Godot source folder to the list of exceptions in your antivirus software.
>
> For Windows Defender, hit the `Windows` key, type "Windows Security" then hit `Enter`. Click on **Virus & threat protection** on the left panel. Under **Virus & threat protection settings** click on **Manage Settings** and scroll down to **Exclusions**. Click **Add or remove exclusions** then add the Godot source folder.

Compiling

Selecting a compiler

SCons will automatically find and use an existing Visual Studio installation. If you do not have Visual Studio installed, it will attempt to use MinGW instead. If you already have Visual Studio installed and want to use MinGW-w64, pass `use_mingw=yes` to the SCons command line. Note

that MSVC builds cannot be performed from the MSYS2 or MinGW shells. Use either `cmd.exe` or PowerShell instead. If you are using MinGW-LLVM, pass both `use_mingw=yes` and `use_llvm=yes` to the SCons command line.

> **Tip**
>
> During development, using the Visual Studio compiler is usually a better idea, as it links the Godot binary much faster than MinGW. However, MinGW can produce more optimized binaries using link-time optimization (see below), making it a better choice for production use. This is particularly the case for the GDScript VM which performs much better with MinGW compared to MSVC. Therefore, it's recommended to use MinGW to produce builds that you distribute to players.
>
> All official Godot binaries are built in custom containers using MinGW.

Running SCons

After opening a command prompt, change to the root directory of the engine source code (using `cd`) and type:

```
C:\godot> scons platform=windows
```

> **Note**
>
> When compiling with multiple CPU threads, SCons may warn about pywin32 being missing. You can safely ignore this warning.

If all goes well, the resulting binary executable will be placed in `C:\godot\bin\` with the name `godot.windows.editor.x86_32.exe` or `godot.windows.editor.x86_64.exe`. By default, SCons will build a binary matching your CPU architecture, but this can be overridden using `arch=x86_64`, `arch=x86_32`, or `arch=arm64`.

This executable file contains the whole engine and runs without any dependencies. Running it will bring up the Project Manager.

> **Tip**

If you are compiling Godot for production use, you can make the final executable smaller and faster by adding the SCons option `production=yes`. This enables additional compiler optimizations and link-time optimization.

LTO takes some time to run and requires up to 30 GB of available RAM while compiling (depending on toolchain). If you're running out of memory with the above option, use `production=yes lto=none` or `production=yes lto=thin` (LLVM only) for a lightweight but less effective form of LTO.

Note

If you want to use separate editor settings for your own Godot builds and official releases, you can enable Self-contained mode by creating a file called `._sc_` or `_sc_` in the `bin/` folder.

Compiling with support for Direct3D 12

By default, builds of Godot do not contain support for the Direct3D 12 graphics API.

To compile Godot with Direct3D 12 support you need at least the following item:

- godot-nir-static library. We compile the Mesa libraries you will need into a static library. Download it anywhere, unzip it and remember the path to the unzipped folder, you will need it below.

Note

You can optionally build the godot-nir-static libraries yourself with the following steps:

1. Install the Python package mako which is needed to generate some files.
2. Clone the godot-nir-static directory and navigate to it.
3. Run the following:

```
git submodule update --init
./update_mesa.sh
scons
```

If you are building with MinGW-w64, add `use_mingw=yes` to the `scons` command, you can also specify build architecture using `arch={architecture}`. If you are building with MinGW-LLVM, add both `use_mingw=yes` and `use_llvm=yes` to the `scons` command.

If you are building with MinGW and the binaries are not located in the `PATH`, add `mingw_prefix="/path/to/mingw"` to the `scons` command.

Mesa static library should be built using the same compiler and the same CRT (if you are building with MinGW) you are using for building Godot.

Optionally, you can compile with the following for additional features:

- PIX is a performance tuning and debugging application for Direct3D12 applications. If you compile-in support for it, you can get much more detailed information through PIX that will help you optimize your game and troubleshoot graphics bugs. To use it, download the WinPixEventRuntime package. You will be taken to a NuGet package page where you can click "Download package" to get it. Once downloaded, change the file extension to .zip and unzip the file to some path.
- Agility SDK can be used to provide access to the latest Direct3D 12 features without relying on driver updates. To use it, download the latest Agility SDK package. You will be taken to a NuGet package page where you can click "Download package" to get it. Once downloaded, change the file extension to .zip and unzip the file to some path.

> **Note**
>
> If you use a preview version of the Agility SDK, remember to enable developer mode in Windows; otherwise it won't be used.

> **Note**
>
> If you want to use a PIX with MinGW build, navigate to PIX runtime directory and use the following commands to generate import library:

```
# For x86-64:
gendef ./bin/x64/WinPixEventRuntime.dll
dlltool --machine i386:x86-64 --no-leading-underscore -d
WinPixEventRuntime.def -D WinPixEventRuntime.dll -l
./bin/x64/libWinPixEventRuntime.a

# For ARM64:
gendef ./bin/ARM64/WinPixEventRuntime.dll
dlltool --machine arm64 --no-leading-underscore -d WinPixEventRuntime.def
-D WinPixEventRuntime.dll -l ./bin/ARM64/libWinPixEventRuntime.a
```

When building Godot, you will need to tell SCons to use Direct3D 12 and where to look for the additional libraries:

```
C:\godot> scons platform=windows d3d12=yes mesa_libs=<...>
```

Or, with all options enabled:

```
C:\godot> scons platform=windows d3d12=yes mesa_libs=<...>
agility_sdk_path=<...> pix_path=<...>
```

Note

For the Agility SDK's DLLs you have to explicitly choose the kind of workflow. Single-arch is the default (DLLs copied to `bin/`). If you pass `agility_sdk_multi_arch=yes` to SCons, you'll opt-in for multi-arch. DLLs will be copied to the appropriate `bin/<arch>/` subdirectories and at runtime the right one will be loaded.

Compiling with ANGLE support

ANGLE provides a translation layer from OpenGL ES 3.x to Direct3D 11 and can be used to improve support for the Compatibility renderer on some older GPUs with outdated OpenGL drivers and on Windows for ARM.

By default, Godot is built with dynamically linked ANGLE, you can use it by placing `libEGL.dll` and `libGLESv2.dll` alongside the executable.

> **Note**
>
> You can use dynamically linked ANGLE with export templates as well, rename aforementioned DLLs to `libEGL.{architecture}.dll` and `libGLESv2.{architecture}.dll` and place them alongside export template executables, and libraries will be automatically copied during the export process.

To compile Godot with statically linked ANGLE:

- Download pre-built static libraries from godot-angle-static library , and unzip them.
- When building Godot, add `angle_libs={path}` to tell SCons where to look for the ANGLE libraries:

```
scons platform=windows angle_libs=<...>
```

> **Note**
>
> You can optionally build the godot-angle-static libraries yourself with the following steps:

1. Clone the godot-angle-static directory and navigate to it.
2. Run the following command:

```
git submodule update --init
./update_angle.sh
scons
```

If you are buildng with MinGW, add `use_mingw=yes` to the command, you can also specify build architecture using `arch={architecture}` . If you are building with MinGW-LLVM, add both `use_mingw=yes` and `use_llvm=yes` to the `scons` command.

If you are building with MinGW and the binaries are not located in the `PATH` , add `mingw_prefix="/path/to/mingw"` to the `scons` command.

ANGLE static library should be built using the same compiler and the same CRT (if you are building with MinGW) you are using for building Godot.

Development in Visual Studio

Using an IDE is not required to compile Godot, as SCons takes care of everything. But if you intend to do engine development or debugging of the engine's C++ code, you may be interested in configuring a code editor or an IDE.

Folder-based editors don't require any particular setup to start working with Godot's codebase. To edit projects with Visual Studio they need to be set up as a solution.

You can create a Visual Studio solution via SCons by running SCons with the `vsproj=yes` parameter, like this:

```
scons platform=windows vsproj=yes
```

You will be able to open Godot's source in a Visual Studio solution now, and able to build Godot using Visual Studio's **Build** button.

See also

See Visual Studio for further details.

Cross-compiling for Windows from other operating systems

If you are a Linux or macOS user, you need to install MinGW-w64 , which typically comes in 32-bit and 64-bit variants, or MinGW-LLVM , which comes as a single archive for all target architectures. The package names may differ based on your distribution, here are some known ones:

Arch Linux	`pacman -Sy mingw-w64`

Debian / Ubuntu	`apt install mingw-w64`
Fedora	`dnf install mingw64-gcc-c++ mingw64-winpthreads-static \` `  mingw32-gcc-c++ mingw32-winpthreads-static`
macOS	`brew install mingw-w64`
Mageia	`urpmi mingw64-gcc-c++ mingw64-winpthreads-static \` `  mingw32-gcc-c++ mingw32-winpthreads-static`

Before attempting the compilation, SCons will check for the following binaries in your `PATH` environment variable:

```
# for MinGW-w64
i686-w64-mingw32-gcc
x86_64-w64-mingw32-gcc

# for MinGW-LLVM
aarch64-w64-mingw32-clang
i686-w64-mingw32-clang
x86_64-w64-mingw32-clang
```

If the binaries are not located in the `PATH` (e.g. `/usr/bin`), you can define the following environment variable to give a hint to the build system:

```
export MINGW_PREFIX="/path/to/mingw"
```

Where `/path/to/mingw` is the path containing the `bin` directory where `i686-w64-mingw32-gcc` and `x86_64-w64-mingw32-gcc` are located (e.g. `/opt/mingw-w64` if the binaries are located in `/opt/mingw-w64/bin`).

To make sure you are doing things correctly, executing the following in the shell should result in a working compiler (the version output may differ based on your system):

```
${MINGW_PREFIX}/bin/x86_64-w64-mingw32-gcc --version
# x86_64-w64-mingw32-gcc (GCC) 13.2.0
```

> **Note**
>
> If you are building with MinGW-LLVM, add `use_llvm=yes` to the `scons` command.

> **Note**
>
> When cross-compiling for Windows using MinGW-w64, keep in mind only `x86_64` and `x86_32` architectures are supported. MinGW-LLVM supports `arm64` as well. Be sure to specify the right `arch=` option when invoking SCons if building from a different architecture.

Troubleshooting

Cross-compiling from some Ubuntu versions may lead to this bug, due to a default configuration lacking support for POSIX threading.

You can change that configuration following those instructions, for 64-bit:

```
sudo update-alternatives --config x86_64-w64-mingw32-gcc
<choose x86_64-w64-mingw32-gcc-posix from the list>
sudo update-alternatives --config x86_64-w64-mingw32-g++
<choose x86_64-w64-mingw32-g++-posix from the list>
```

And for 32-bit:

```
sudo update-alternatives --config i686-w64-mingw32-gcc
<choose i686-w64-mingw32-gcc-posix from the list>
sudo update-alternatives --config i686-w64-mingw32-g++
<choose i686-w64-mingw32-g++-posix from the list>
```

Creating Windows export templates

Windows export templates are created by compiling Godot without the editor, with the following flags:

```
C:\godot> scons platform=windows target=template_debug arch=x86_32
C:\godot> scons platform=windows target=template_release arch=x86_32
C:\godot> scons platform=windows target=template_debug arch=x86_64
C:\godot> scons platform=windows target=template_release arch=x86_64
C:\godot> scons platform=windows target=template_debug arch=arm64
C:\godot> scons platform=windows target=template_release arch=arm64
```

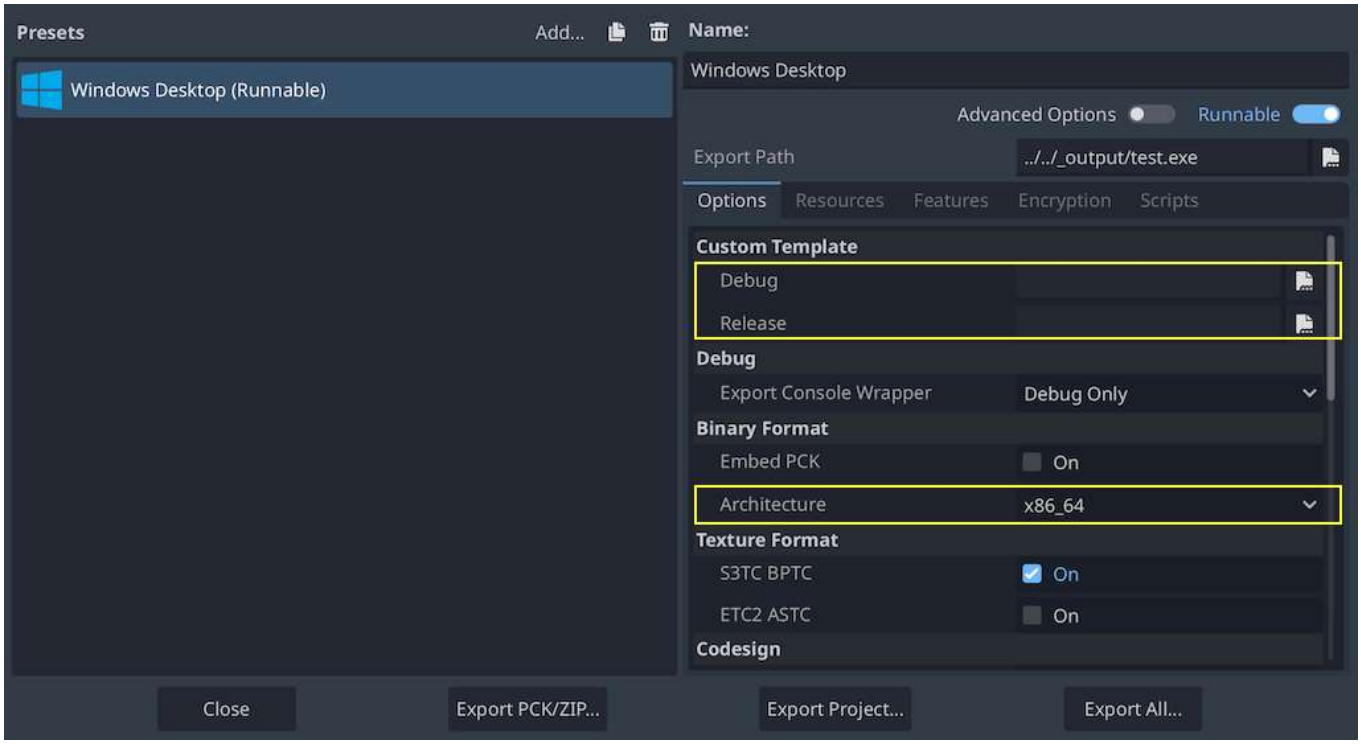
If you plan on replacing the standard export templates, copy these to the following location, replacing `<version>` with the version identifier (such as `4.2.1.stable` or `4.3.dev`):

```
%APPDATA%\Godot\export_templates\<version>\
```

With the following names:

```
windows_debug_x86_32_console.exe
windows_debug_x86_32.exe
windows_debug_x86_64_console.exe
windows_debug_x86_64.exe
windows_debug_arm64_console.exe
windows_debug_arm64.exe
windows_release_x86_32_console.exe
windows_release_x86_32.exe
windows_release_x86_64_console.exe
windows_release_x86_64.exe
windows_release_arm64_console.exe
windows_release_arm64.exe
```

However, if you are using custom modules or custom engine code, you may instead want to configure your binaries as custom export templates here:



Select matching architecture in the export config.

You don't need to copy them in this case, just reference the resulting files in the `bin\` directory of your Godot source folder, so the next time you build, you will automatically have the custom templates referenced.

User-contributed notes

Please read the User-contributed notes policy before submitting a comment.

3 comments · 3 replies – *powered by giscus*

Oldest Newest

stokatyan Jan 18

I think the following line is incorrect:

```
If all goes well, the resulting binary executable will be placed in C:\godot\bin\
```

Shouldn't it say the binary should be in the root directory of the engine source, which is where the scons command is run?

↑ 1 ☺ 2 replies

akien-mga Jan 18 Member

The line is correct. Earlier in the page there is this statement:

> The tutorial will assume from now on that you placed the source code in `C:\godot`.

And then generated files indeed go in the `bin` folder of the Godot source folder, i.e. `C:\godot\bin` with the given assumption.

☺ 👍 1

stokatyan Jan 18

You are correct, I missed that. Thanks for pointing it out!

☺

cjmaxik Aug 16

If you're using Windows 10/11 Pro with Hyper-V enabled, you can have a clean build machine spawned relatively quickly.

1. In Hyper-V, click "Quick Create..."
2. Select and create "Windows 11 dev environment"
3. When created and connected to, install Git, Python and SCons as you wish, for example:

```
winget install Git.Git
winget install Python.Python.3.12
pip install SCons
```

4. ...
5. PROFIT!

↑ 1 ☺ 0 replies

0 replies

Gromph 16 days ago

Where do you get libEGL.dll and libGLESv2.dll?

↑ 1 

1 reply

bruvzg 14 days ago Member edited

Normally Godot uses statically linked ANGLE, there are no DLLs, you can download static libraries from https://github.com/godotengine/godot-angle-static/releases

If you want to link it dynamically, you can either build ANGLE yourself or get pre-built libraries from CEF distribution (download Minimal Distribution for target platform), there are no official builds.

Write Preview Aa

Sign in to comment