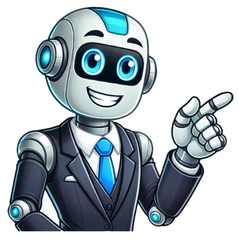
Click Here



The build process can be customized using environment variables, but there are specific rules for single-config and multi-config generators. For single-config generators like Unix Makefiles or Ninja, the CMAKE_BUILD_TYPE variable is used to specify the build type, which can be set in the configure command. In contrast, multi-config generators such as Visual Studio generators and Ninja Multi-Config require specifying configurations during the configuration command via CMAKE_CONFIGURATION_TYPES. The config to build is specified in the build command using the --config argument. For instance, in CMake 3.17, a default value can be set for the Ninja Multi-Config generator by setting the CMAKE_DEFAULT_BUILD_TYPE variable, which will be used if the --config argument isn't passed to the build command. On the other hand, Visual Studio generators do not support specifying a default configuration at configure time. ###ARTICLESolved: Xcopy.exe Issues in Visual Studio Post-Build Events To resolve the issue with xcopy.exe in post-build events of Visual Studio, follow these steps: Looking forward to seeing everyone at the meeting tomorrow, where we'll discuss our strategies in detail. For me, when copying XCopy command from web into VS, sometimes it caused a problem. To know what is the missing file, you can go to Output window and it will show you straight away what went missing. I fixed this by simply restarting Visual Studio - I had just run dotnet tool install xxx in a console window and VS hadn't yet picked up the new environment variables and/or path settings that were changed, so a quick restart fixed the issue. For some, disk space was low, and files that couldn't be written were expected to be present later. Other answers mentioned missing files (or misnamed/improperly referenced-by-name files)--but the root cause was lack of disk space. I've got compiler errors for projects that have commands in PreBuild events. 'cmd' is not recognized as an internal or external command, operable program or batch file. When I closed Visual Studio and opened it again, the problem was fixed. Yet another variant of file not found, because of spaces in the path. In my case in the msbuild script. I needed to use HTML style " strings within the exec command. Application use Command line arguments, I removed them and then added them back. Suddenly the project failed to build. Verify that you use 'Debug' tab (not 'Build Events' tab) -> Command Line Arguments. My solution was just simple as: have you tried turning it off and on again? So I restarted the computer and the issue was gone. If development environment is windows and visual studio project is on C: drive.. Then make sure that visual studio is run with administrator right.. simply right click and 'Run as administrator'. You can also go to properties of visual studio project -> Advance -> and enable 'Run as administrator'. I had similar issue and I received: The command which was caused by my post build script. Finally I found out the root cause is I did not populate the missing information in the .nuspec file, i.e. replacing all the variables between $ and $ with the actual value, e.g. replacing $author$ with my name Check the Output tab carefully. That should reveal the issue reason. I actually got this error message when signtool.exe itself could not be found. To fix this I added the folder path to the PATH environment variable and restarted Visual Studio. 1 I had similar issue and I received: The command Build and Rebuild are two commands used to manage dependencies in a project, but they serve different purposes and have different consequences. When building a solution, the build command generates all intermediate and output files that are out-of-date with respect to their source files. However, this process can fail due to bugs or issues with Visual Studio's out-of-date evaluation system, leading to errors. In such cases, rebuilding the project is often used as a workaround to create new intermediate files and hope for the best. The clean command is also used in some scenarios, although its implementation can be hacky and incomplete. To determine when to use each command, one needs to understand what they do and consider their usage based on personal preferences or circumstances. Some people prefer using build most of the time and only resorting to rebuild if needed, while others tend to use rebuild more frequently due to its reliability. The problem with current Visual Studio UX is that it relies too heavily on these two commands, without providing a clear or intuitive solution for users to resolve issues. Microsoft should consider addressing this issue by fixing build, improving clean, and eliminating the need for rebuild in certain situations. For example, in a multi-module Maven project, using both build and test dependencies can lead to complications during deployment. Within a csproj, for example, mySolution.sln - my.project - my.project.csproj - Foo.cs - sql - create.sql - merge.sql - delete.sql, you can use to specify the copying behavior of SQL files. For instance: PreserveNewest Or, when outside the projects, for example, mySolution.sln - my.project - my.project.csproj - Foo.cs - sql - orders - create.sql - merge.sql - delete.sql, you can use: sql\%(RecursiveDir)%(Filename)%(Extension) PreserveNewest Alternatively, structuring the content folder with a subdirectory representing the parent directory in your bin output can also help. For example: mySolution.sln - my.project - my.project.csproj - Foo.cs - sqlcontent - sql - orders - create.sql - merge.sql - delete.sql Then you would use: PreserveNewest With the new multi-stage pipeline, you typically utilize the first stage to construct your artifacts, followed by subsequent stages for deployment - an analogous process to before but condensed into a single module. For those who have employed the build & release pipeline in the past, you'll notice the old build definition within the new Pipeline module and the old release definition in the older Release module. Notably, YAML-based release pipelines were never adopted due to their intended obsolescence with the introduction of multi-stage pipelines. In light of this transition, it's recommended that users abandon classic Release Pipelines in favor of the newer "Pipeline" module. The Roslyn C# compiler platform operates under .NET Standard 2.0, allowing compatibility with both .NET Framework 4.6.1+ and .NET Core 2.0+(1). Visual Studio, which incorporates MSBuild, runs on .NET Framework, whereas dotnet build functions as a standalone .NET Core application. The latter is equipped to handle SDK-style csprojs exclusively, leveraging Roslyn on .NET Core through its integration with the .NET Core SDKs. Both methods - utilizing Visual Studio or MSBuild - invoke the same compiler code but differ in terms of execution environment and compatibility. dotnet build presents an advantage when employed from the command line, whereas Visual Studio excels within the .NET Framework and Windows-only domain. It's worth acknowledging that Roslyn's runtime environment does not influence the compiled IL, which can be successfully executed by both .NET Core and .NET Framework-based configurations. Additionally, analyzers targeting .NET Core are limited to dotnet build due to their reliance on specific target frameworks.