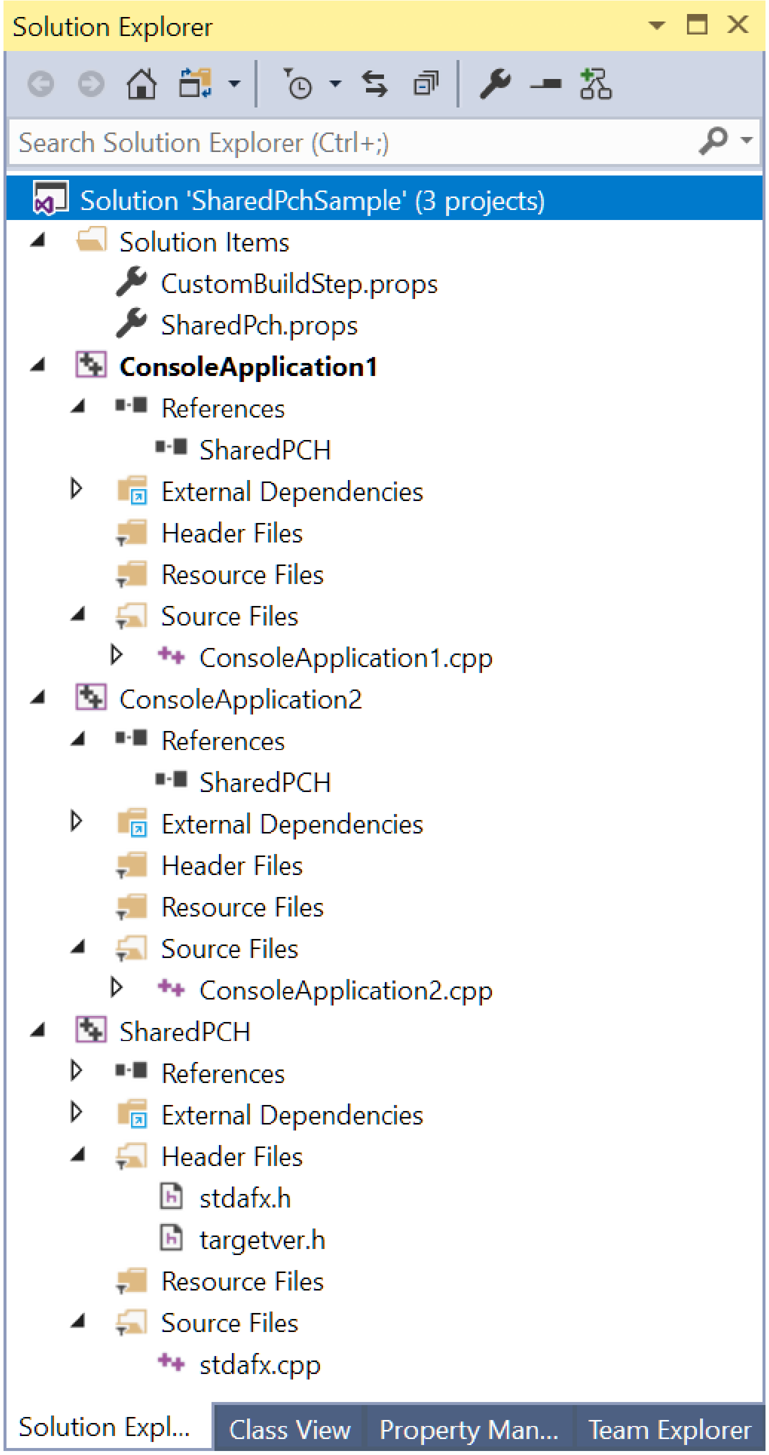
Shared PCH usage sample

Often many projects in the solution use the same or very similar precompiled headers. As pch files are usually big and building them takes significant time, it is a popular question - can several projects use the same pch file which would be built just once?

The answer is yes, but it requires couple of tricks to satisfy cl's check that command line used for building pch is the same as command line used for building a source file using this pch.

Here is a sample solution, which has 3 projects – one (SharedPCH) is building the pch and the static library and the other two (ConsoleApplication 1 and 2) are using it.



```
Solution 'SharedPchSample' (3 projects)
  Solution Items
    CustomBuildStep.props
    SharedPch.props
  ConsoleApplication1
    References
      SharedPCH
    External Dependencies
    Header Files
    Resource Files
    Source Files
      ConsoleApplication1.cpp
  ConsoleApplication2
    References
      SharedPCH
    External Dependencies
    Header Files
    Resource Files
    Source Files
      ConsoleApplication2.cpp
  SharedPCH
    References
    External Dependencies
    Header Files
      stdafx.h
      targetver.h
    Resource Files
    Source Files
      stdafx.cpp
```

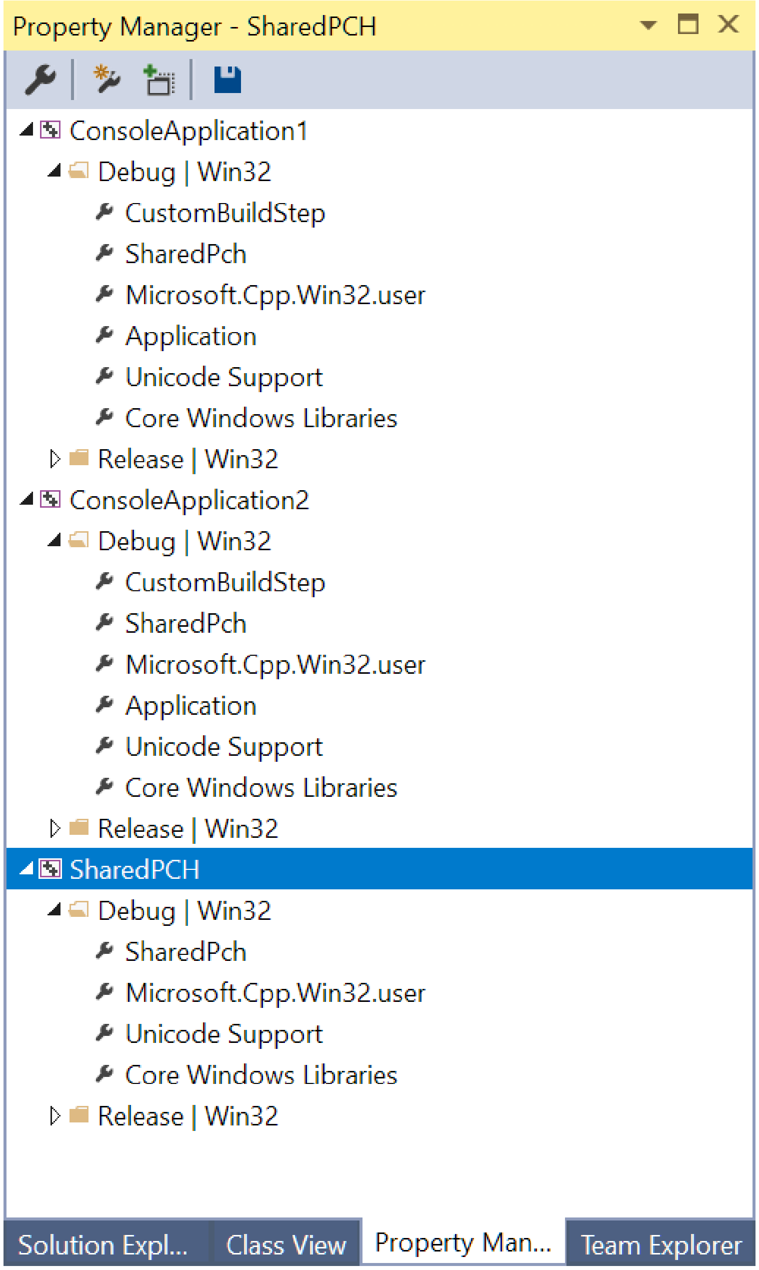
When ConsoleApplication projects reference the SharedPCH one, the build will automatically link the SharedPCH's static lib. But several project properties need to be changed as well.

1. C/C++ Additional Include Directories (/I) should contain the shared stdafx.h directory
2. C/C++ Precompiled Header Output File (/Fp) should be set to the shared pch file (produced by SharedPCH project)

3. If your projects are compiled with /Zi or /ZI (see more info about those switches at the end), the projects which use shared pch need to copy .pdb and .idb files produced by shared pch project to their specific locations so final pdb contains pch symbols.

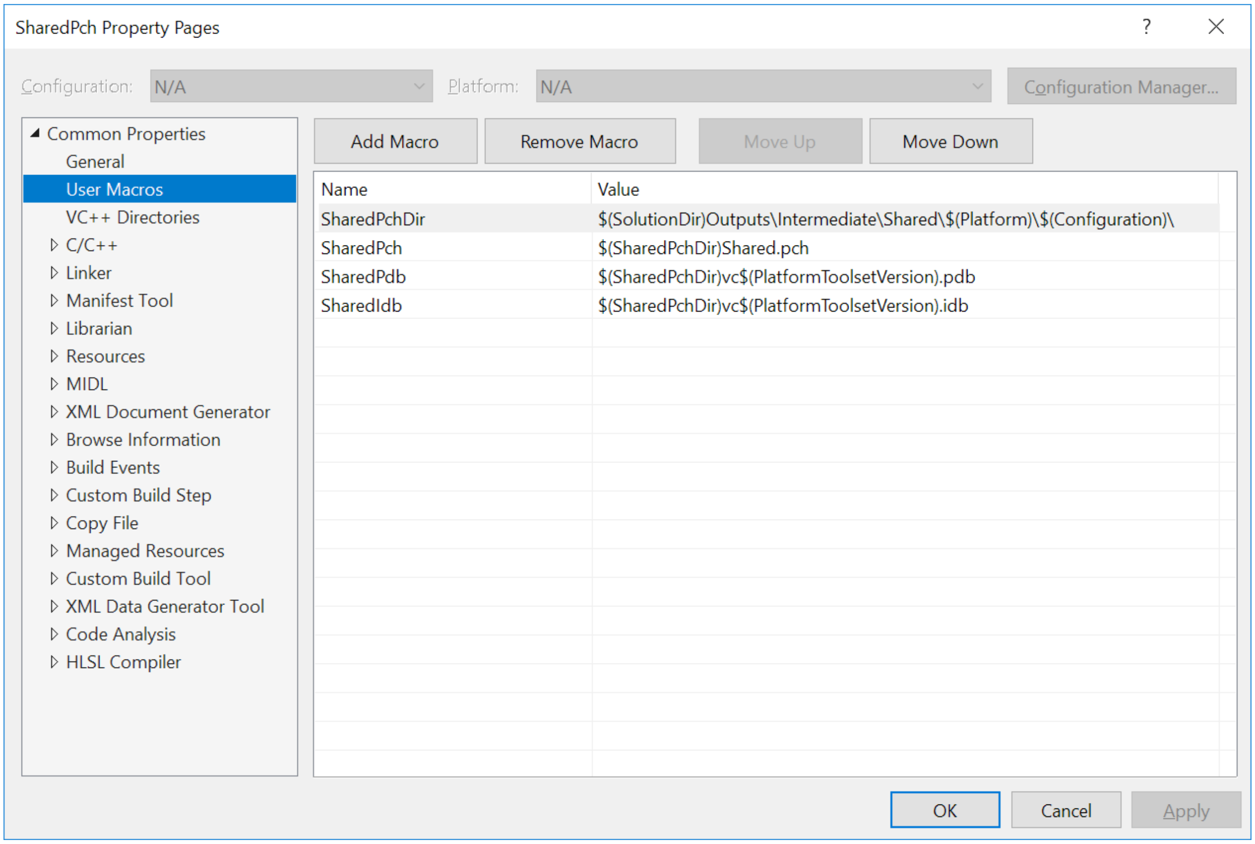
As those properties need to be changed similarly for all projects, I created the SharedPCH.props and CustomBuildStep.props files and imported them to my projects using Property Manager toolwindow.

SharedPch.props helps with #1 and #2 and is imported in all projects. CustomBuildStep.props helps with #3 and imported to consuming pch projects, but not to the producing one. If your projects are using /Z7, you don't need CustomBuildStep.props.



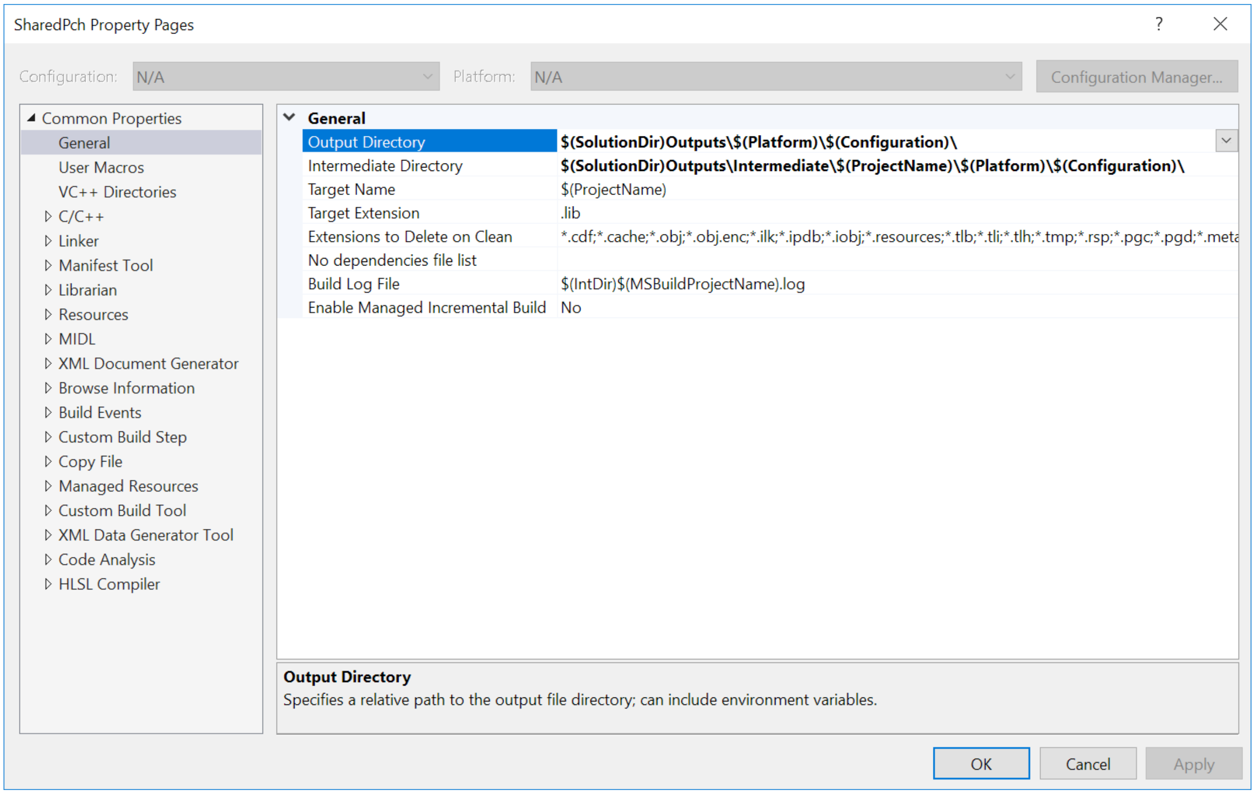
```
◢ ConsoleApplication1
    ◢ Debug | Win32
        CustomBuildStep
        SharedPch
        Microsoft.Cpp.Win32.user
        Application
        Unicode Support
        Core Windows Libraries
    ▷ Release | Win32
◢ ConsoleApplication2
    ◢ Debug | Win32
        CustomBuildStep
        SharedPch
        Microsoft.Cpp.Win32.user
        Application
        Unicode Support
        Core Windows Libraries
    ▷ Release | Win32
◢ SharedPCH
    ◢ Debug | Win32
        SharedPch
        Microsoft.Cpp.Win32.user
        Unicode Support
        Core Windows Libraries
    ▷ Release | Win32
```

In SharedPch.props I defined properties for shared pch, pdb and idb files locations:



Name	Value
SharedPchDir	$(SolutionDir)Outputs\Intermediate\Shared\$(Platform)\$(Configuration)\
SharedPch	$(SharedPchDir)Shared.pch
SharedPdb	$(SharedPchDir)vc$(PlatformToolsetVersion).pdb
SharedIdb	$(SharedPchDir)vc$(PlatformToolsetVersion).idb

I wanted to have all build outputs under one root folder, separate from the sources, so I redefined Output and Intermediate directories - this is not necessary for using shared pch, just makes experimentations easier as you can delete one folder if something goes wrong.



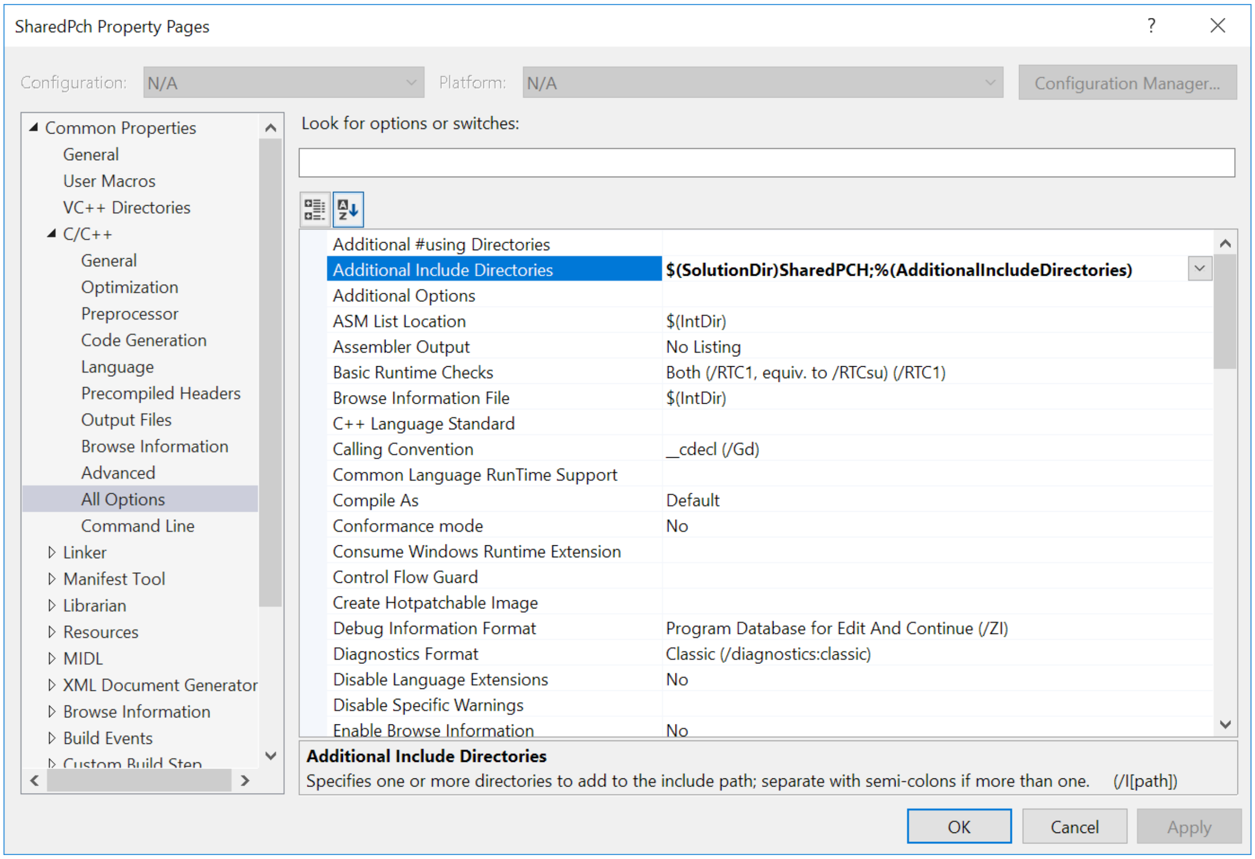
General

Output Directory	**$(SolutionDir)Outputs\$(Platform)\$(Configuration)**
Intermediate Directory	**$(SolutionDir)Outputs\Intermediate\$(ProjectName)\$(Platform)\$(Configuration)**
Target Name	$(ProjectName)
Target Extension	.lib
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.obj.enc;*.ilk;*.ipdb;*.iobj;*.resources;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;*.meta
No dependencies file list	
Build Log File	$(IntDir)$(MSBuildProjectName).log
Enable Managed Incremental Build	No

Output Directory

Specifies a relative path to the output file directory; can include environment variables.

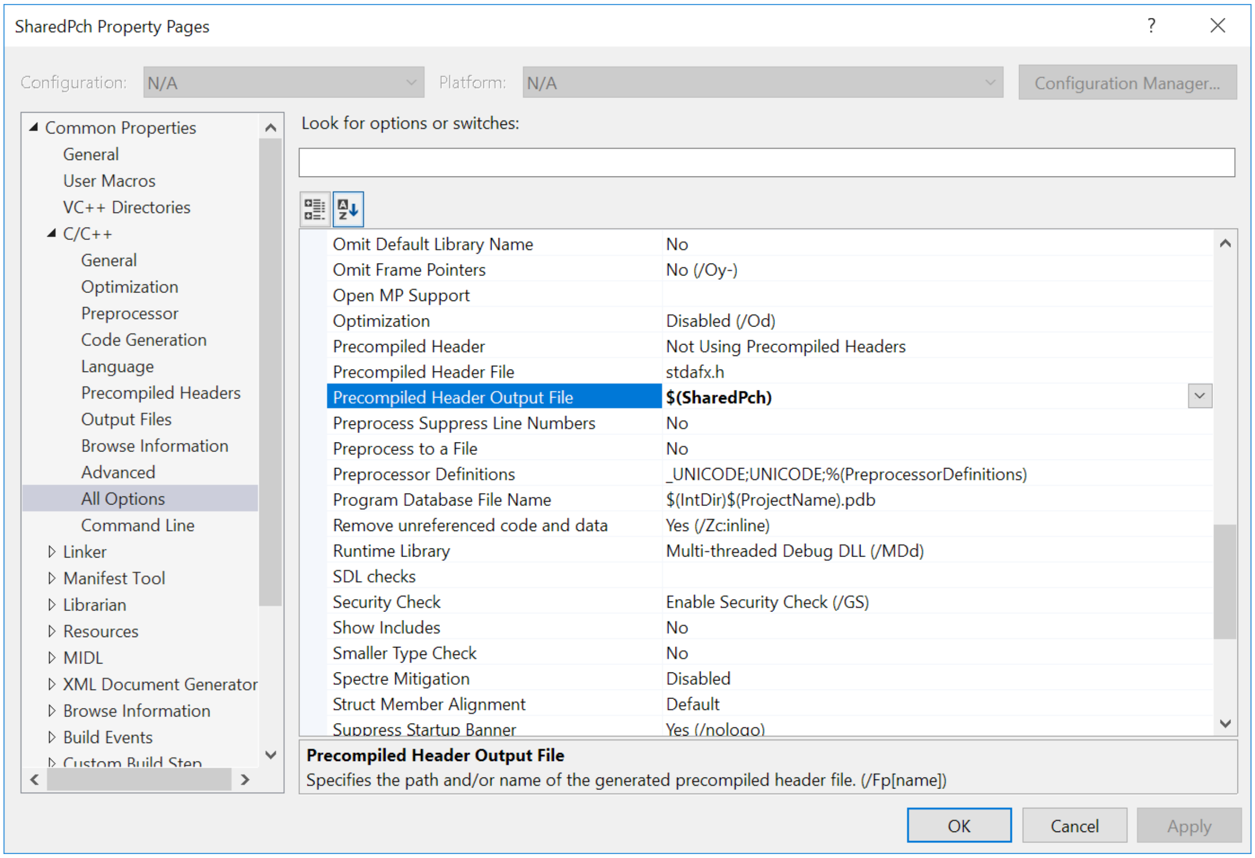
And adjusted C/C++ ‘Additional Include Directories’ and ‘Precompiled Header Output File’ properties:



Additional #using Directories	
Additional Include Directories	**$(SolutionDir)SharedPCH;%(AdditionalIncludeDirectories)**
Additional Options	
ASM List Location	$(IntDir)
Assembler Output	No Listing
Basic Runtime Checks	Both (/RTC1, equiv. to /RTCsu) (/RTC1)
Browse Information File	$(IntDir)
C++ Language Standard	
Calling Convention	__cdecl (/Gd)
Common Language RunTime Support	
Compile As	Default
Conformance mode	No
Consume Windows Runtime Extension	
Control Flow Guard	
Create Hotpatchable Image	
Debug Information Format	Program Database for Edit And Continue (/ZI)
Diagnostics Format	Classic (/diagnostics:classic)
Disable Language Extensions	No
Disable Specific Warnings	
Enable Browse Information	No

Additional Include Directories

Specifies one or more directories to add to the include path; separate with semi-colons if more than one. (/I[path])



Omit Default Library Name	No
Omit Frame Pointers	No (/Oy-)
Open MP Support	
Optimization	Disabled (/Od)
Precompiled Header	Not Using Precompiled Headers
Precompiled Header File	stdafx.h
Precompiled Header Output File	**$(SharedPch)**
Preprocess Suppress Line Numbers	No
Preprocess to a File	No
Preprocessor Definitions	_UNICODE;UNICODE;%(PreprocessorDefinitions)
Program Database File Name	$(IntDir)$(ProjectName).pdb
Remove unreferenced code and data	Yes (/Zc:inline)
Runtime Library	Multi-threaded Debug DLL (/MDd)
SDL checks	
Security Check	Enable Security Check (/GS)
Show Includes	No
Smaller Type Check	No
Spectre Mitigation	Disabled
Struct Member Alignment	Default
Suppress Startup Banner	Yes (/nologo)

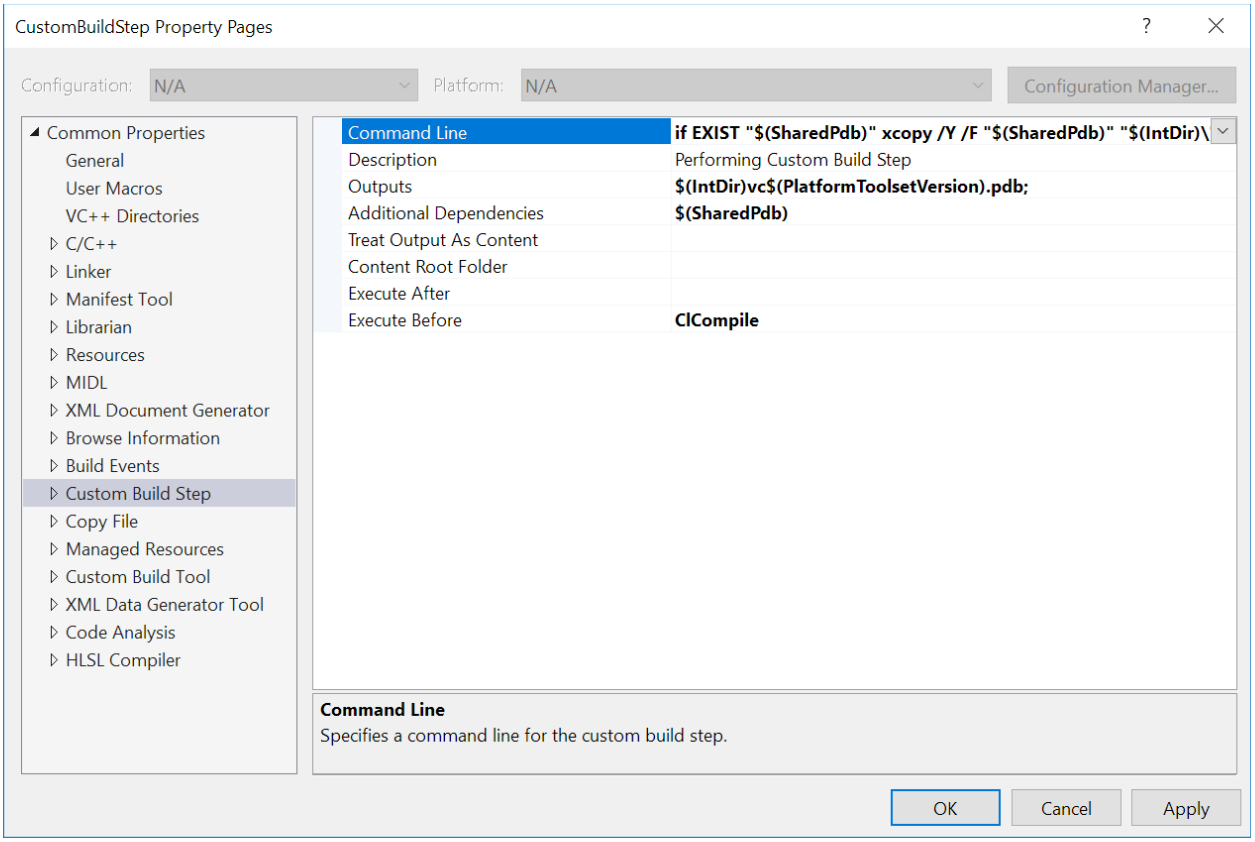
Precompiled Header Output File

Specifies the path and/or name of the generated precompiled header file. (/Fp[name])

In CustomBuildStep.props I defined Custom Build Step to run before ClCompile target and copy shared pch .pdb and .idb files if they are newer than the project's .pdb and .idb files. Note, that we are talking about compiler intermediate pdb file here, not the final one produced by the linker.

If all files in the project are using one pch – that all we need to do as when pch is changed, all other files need to be recompiled as well, so at the end of the build we'll have full pdb and idb files.

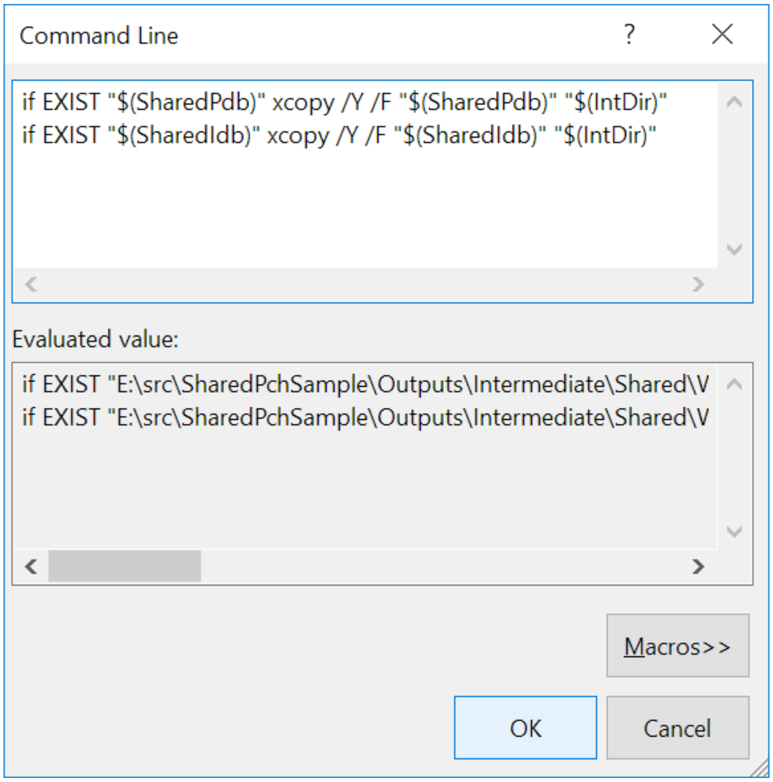
If your project uses more than one pch or contains files that are not using pch at all, you'll need to change pdb file location (/Fd) for those files, so it is not overridden by the shared pch pdb.



Command Line	if EXIST "$(SharedPdb)" xcopy /Y /F "$(SharedPdb)" "$(IntDir)\
Description	Performing Custom Build Step
Outputs	$(IntDir)vc$(PlatformToolsetVersion).pdb;
Additional Dependencies	$(SharedPdb)
Treat Output As Content	
Content Root Folder	
Execute After	
Execute Before	ClCompile

Command Line
Specifies a command line for the custom build step.

I used the command line property editor to define the commands. Each 'xcopy' command should be on its own line there:



```
if EXIST "$(SharedPdb)" xcopy /Y /F "$(SharedPdb)" "$(IntDir)"
if EXIST "$(SharedIdb)" xcopy /Y /F "$(SharedIdb)" "$(IntDir)"
```

```
if EXIST "E:\src\SharedPchSample\Outputs\Intermediate\Shared\V
if EXIST "E:\src\SharedPchSample\Outputs\Intermediate\Shared\V
```

Alternatively, you can put all commands in a script file and just specify it as command line.

Background information

/Z7, /ZI and /Zi compiler flags

When /Z7 is used, the debug information (mainly type information) is stored in each OBJ file. This includes types from the header files, which means that there is a lot of duplication in case of shared headers and OBJ size can be huge.

When /Zi or /ZI is used, the debug information is stored in a compiler PDB file. In a project, the source files often use the same PDB file (this is controlled by /Fd compiler flag, the default value is $(IntDir)vc$(PlatformToolsetVersion).pdb), so the debug information is shared across them.

/ZI will also generate an IDB file to store information related to incremental compilation to support Edit and Continue.

Compiler PDB vs. linker PDB

As mentioned above, the compiler PDB is generated by /Zi or /ZI to store debug information. Later, linker will generate a linker PDB by combining the information from the compiler PDB and additional debug information during linking. Linker may also remove unreferenced debug information. The name of the linker PDB is controlled by /PDB linker flag, the default value is $(OutDir)$(TargetName).pdb.