How config CLANG Compiler (32 and 64 bits) (with MSVC) into CodeBlocks

Full name of tutorial : How config CLANG Compiler (32 and 64 bits) (+ MSVC +SDK Windows) into Code::Blocks on Windows 11 64 bits.

```
Code::Blocks : the best and great free IDE for Windows, Linux and ... Mac OS
```

Presentation of CLANG/LLVM Compiler (32 and 64 bits) [+ MSVC + SDK Windows].

During first run of CB on Windows, this IDE detect automatically some compilers, or present one list of them pre-configured.
It's very good functionnality, but, sometimes, you must "force" these configurations proposed by default to run correctly.
This tuto describe how configure one major compiler C\C++ on Windows : CLANG/LLVM Compiler (32 and 64 bits) associated with MSVC (Visual Studio Community 2022) + SDK Windows.

CLANG is rigourously a C/C++ (very good) compiler without libraries or "include files" needed to success generation.
On Windows systems, it's mandatory to associate CLANG with another development environment like MSVC + SDK Windows, or MinGW32/64 (most possibilities ...).

Installation of CLANG/LLVM Compiler (32 and 64 bits) [+ MSVC + SDK Windows.]

I suppose that you have installed before Visual Studio Community 2022 and SDK Windows 11 on your system Windows 11 64 bits.
This package is available on site : https://github.com/llvm/llvm-project/releases

Two files are needed to download in version 32 and 64 bits :
"llvm-18.1.8-win64.exe" (64 bits) and
"llvm-18.1.8-win32.exe" (32 bits)

After download, click on these files to install resuired components on your system. Second installation detect first, and you must (if you want to try two versions 32 bits and 64 bits on your system) choose to "conserve" precedent installation.
- LLVM + CLANG 32 bits is installed on directory "C:\Program Files (x86)\LLVM" by default.
- LLVM + CLANG 64 bits is installed on directory "C:\Program Files\LLVM" by default.

On these two directories, you can use tool "Uninstall.exe" to ... uninstall all components ... -)
It's important, because if you upgrade with superior version, you must uninstall before, and only one version of CLANG/LLVM can be uninstalled using application manager included in "Parameters" into Windows systems. Think to desinstall second with tool "uninstall".

Normally, after these installations, next run of IDE Code::Blocks must detect CLANG/LLVM compiler and propose it into list of available compilers on system. Search if you find "CLANG/LLVM Compiler" in this list. Verify what version is detected ?
If not, apply, next instructions.
First, you must access into IDE CB at menu "Settings" and submenu "Compilers" to select "Visual Studio 2022".
You must first copy it and rename this ident of compiler in "LLVM Clang X64 (64 bits)" (by example).

An after, you must verify field "Compiler's installation directory" that must contain
- "C:\Program Files\LLVM\bin"

You must verify list of tools like this (in tab "Program Files") :
- C Compiler : clang.exe
- C++ Compiler : clang++.exe
- Linker for dynamic libs : lld-link.exe
- Linker for statics libs : llvm-lib.exe
- Debugger :
- Resource compiler : llvm-rc.exe
- Make program : nmake.exe (this included in Visual Studio Build Tools 2002)

After, you must add in tab "Additional Paths" :
- "C:\Program Files (x86)\Windows Kits\10\bin\10.0.22621.0\x64" (by example)
- and : "C:\Program Files (x86)\Windows Kits\10\bin\x64"
- and : "C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\MSVC\14.40.33807\bin\Hostx64\x64"
- and : "C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Current\Bin\amd64"

WARNING : After update, all number included into name of directories of MSVC/SDK can change !!!
To integrate these evolutions, it's seem rational to define environment variables on system like

this :
- VS_VERSION define to 2022 (at date)
- VS_NUM define to 14.40.33807 (at date) and
- KIT_VERSION define to 10 (at date)
- KIT_NUM define to 10.0.22621.0 (at date)

Then, you can use these variables into configuration of CB with use of %var% into directory name to be independant of evolutions.

And, it's not all, you must verify in tab "Search Directories" and select "Compiler", or "Linker" or "Resource Compiler" subtabs.

For compiler, search directories of "include files" are (and it's the same for "Resource Compiler") :
- C:\Program Files\LLVM\lib\clang\18\include
- C:\Program Files\Microsoft Visual Studio%VS_VERSION%\Community\VC\Tools\MSVC%VS_NUM%\include
- C:\Program Files\Microsoft Visual Studio%VS_VERSION%\Community\VC\Auxiliary\VS\include
- C:\Program Files (x86)\Windows Kits%KIT_VERSION%\Include%KIT_NUM%\ucrt
- C:\Program Files (x86)\Windows Kits%KIT_VERSION%\Include%KIT_NUM%\um
- C:\Program Files (x86)\Windows Kits%KIT_VERSION%\Include%KIT_NUM%\shared
- C:\Program Files (x86)\Windows Kits%KIT_VERSION%\Include%KIT_NUM%\winrt
- C:\Program Files (x86)\Windows Kits%KIT_VERSION%\Include%KIT_NUM%\cppwinrt

For linker, search directories of "lib files" are :
- C:\Program Files\LLVM\lib\clang\18\lib\windows
- C:\Program Files\Microsoft Visual Studio%VS_VERSION%\Community\VC\Tools\MSVC%VS_NUM%\lib\x64
- C:\Program Files (x86)\Windows Kits%KIT_VERSION%\Lib%KIT_NUM%\ucrt\x64
- C:\Program Files\Microsoft Visual Studio%VS_VERSION%\Community\VC\Tools\MSVC%VS_NUM%\lib\x64\store
- C:\Program Files (x86)\Windows Kits%KIT_VERSION%\Lib%KIT_NUM%\um\x64

Remark : If you wan't use environment variables, you can translate these by "real value" of directory into CB, shame.

To terminate with 64 bits version of CLANG/LLVM, you must select in tab "Compiler settings" (or in zone "Other Compiler options") : "-m64". CLANG/LLVM must position automatically this option, but it's a precaution.

It's some tedious, but with these "full" configurations, you can build an program on Windows 11 64 bits with success.

Result of command "C:\Program Files\LLVM\bin\clang.exe" --version, must be :
clang version 18.1.8
Target: x86_64-pc-windows-msvc
Thread model: posix
InstalledDir: C:\Program Files\LLVM\bin

Now, you must copy "LLVM Clang X64 (64 bits)" in list of available compilers into CB (menu "Settings" submenu "Compilers"), and rename it by "LLVM Clang X32 (32 bits)" by example.

New configurations for this compiler in version 32 bits are next :

In tab "Toolchain executable", you must change value by : "C:\Program Files (x86)\LLVM\bin"

In subtab "Additional Paths" :
- C:\Program Files (x86)\Windows Kits%KIT_VERSION%\bin%KIT_NUM%\x86
- C:\Program Files (x86)\Windows Kits%KIT_VERSION%\bin\x86
- C:\Program Files\Microsoft Visual Studio%VS_VERSION%\Community\VC\Tools\MSVC%VS_NUM%\bin\Hostx86\x86
- C:\Program Files\Microsoft Visual Studio%VS_VERSION%\Community\MSBuild\Current\Bin\amd64

In tab "Search Paths", changes are mandatory in subtab "Compiler" and "Resource compiler" : replace
- "C:\Program Files\LLVM\lib\clang\18\include" by "C:\Program Files (x86)\LLVM\lib\clang\18\include"

And, changes in tab "Linker" are next :
- C:\Program Files (x86)\LLVM\lib\clang\18\lib\windows
- C:\Program Files\Microsoft Visual Studio%VS_VERSION%\Community\VC\Tools\MSVC%VS_NUM%\lib\x86
- C:\Program Files (x86)\Windows Kits%KIT_VERSION%\Lib%KIT_NUM%\ucrt\x86
- C:\Program Files\Microsoft Visual Studio%VS_VERSION%\Community\VC\Tools\MSVC%VS_NUM%\lib\x86\store
- C:\Program Files (x86)\Windows Kits%KIT_VERSION%\Lib%KIT_NUM%\um\x86

And, to terminate with 32 bits version of CLANG/LLVM, you must select in tab "Compiler settings" (or in zone "Other Compiler options") : "-m32". CLANG/LLVM must position automatically this option, but it's a precaution.

Test of "simple" code with CLANG + MSVC + SDK Windows into CB

And, with simply source "hellowworld.c", you can test generation of program into IDE CB, choosing "create new project" in main windows of CB, and choose "console application" with no source proposed by default, because named "main.c" by default, and choose compiler "LLVM Clang X64 (64 bits)".
You can select good directory/source with option "add file" after first creation of project into CB.

One time project created, you can generate it with selecting main menu "Build" and choose submenu "Rebuild..." (or CTRL-F11).

Pleasure of programming is open for you, your imagination is illimited, at your keyboard ! Enjoy !

PS : source file "hellowworld.c" :

/ Basic example in language C : hellowworld.c /

```
#include <stdio.h>

int main(int argc, char argv[]) {
/ printf() displays the string inside quotation */
printf("Hello, World!");
return 0;
}
```

PS2 : Use of CLANG compiler + MSVC + SDK Windows in command line (just to illustrate)

You can also use CLANG compiler + MSVC+ SDK Windows in command console on Windows (CMD.EXE) with next instructions :

```
set PATHSAV=%PATH%
set INCSAV=%INCLUDE%
set LIBPSAV=%LIBPATH%
set PATH=C:\Program Files\LLVM\bin;C:\Program Files\Microsoft Visual
Studio%VS_VERSION%\Community\VC\Tools\MSVC%VS_NUM%\bin\Hostx64\x64;C:\Progra
m Files (x86)\Windows Kits%KIT_VERSION%\bin%KIT_NUM%\x64;%PATH%
set PATH=C:\Program Files (x86)\Windows Kits%KIT_VERSION%\bin\x64;C:\Program
Files\Microsoft Visual
Studio%VS_VERSION%\Community\MSBuild\Current\Bin\amd64;%PATH%
set INCLUDE=C:\Program Files\Microsoft Visual
Studio%VS_VERSION%\Community\VC\Tools\MSVC%VS_NUM%\include;C:\Program
```

```
Files\Microsoft Visual Studio%VS_VERSION%\Community\VC\Auxiliary\VS\include;C:\Program
Files (x86)\Windows Kits%KIT_VERSION%\Include%KIT_NUM%\ucrt;
set INCLUDE=C:\Program Files\LLVM\lib\clang\18\include;C:\Program Files (x86)\Windows
Kits%KIT_VERSION%\Include%KIT_NUM%\um;C:\Program Files (x86)\Windows
Kits%KIT_VERSION%\Include%KIT_NUM%\shared;C:\Program Files (x86)\Windows
Kits%KIT_VERSION%\Include%KIT_NUM%\winrt;%INCLUDE%
set LIB="C:\Program Files\Microsoft Visual
Studio%VS_VERSION%\Community\VC\Tools\MSVC%VS_NUM%\lib\x64";"C:\Program
Files\Microsoft Visual
Studio%VS_VERSION%\Community\VC\Tools\MSVC%VS_NUM%\lib\x64\store"
set LIB="C:\Program Files\LLVM\lib\clang\18\lib\windows";"C:\Program Files (x86)\Windows
Kits%KIT_VERSION%\Lib%KIT_NUM%\ucrt\x64";"C:\Program Files (x86)\Windows
Kits%KIT_VERSION%\Lib%KIT_NUM%\um\x64";%LIB%
```

```
REM Compile + link in one pass
clang hellowworld.c -o hellowworld.exe
REM Compile + link in two pass
clang -c hellowworld.c -o hellowworld.obj
lld-link.exe hellowworld.obj /out:hellowworld.exe /machine:X64 /subsystem:console
/libpath:"C:\Program Files (x86)\Windows Kits%KIT_VERSION%\Lib%KIT_NUM%\um\x64"
/Libpath:"C:\Program Files\LLVM\lib\clang\18\lib\windows" /libpath:"C:\Program Files\Microsoft
Visual Studio%VS_VERSION%\Community\VC\Tools\MSVC%VS_NUM%\lib\x64" \
/libpath:"C:\Program Files (x86)\Windows Kits%KIT_VERSION%\Lib%KIT_NUM%\ucrt\x64"
kernel32.lib user32.lib libcmt.lib
```

Continue with use of CLANG compiler, and don't forgive, at the end of your work, to return to initial state :

```
set PATH=%PATHSAV%
set INCLUDE=%INCSAV%
set LIB=%LIBSAV%
```

And, with precedent example, you can also generate version 32 bits with "clang" but, you must change values of PATH, INCLUDE and LIB
with good directories described before (don't forgive to change "/machine:X86" during call of "link" and /libpath:"...." if two pass).

But, it's much easy to use CLANG/LLVM associated with MSVC +SDK directly into CB IDE especially with complex C program (many C sources and many subdirectories ...) , and multiple targets by example : main DLL and console program to test this DLL, ... -)

PS3 : Syntax of tools CLANG/LLVM

Syntax and list of options of CLANG/LLVM compiler "clang" is very ... long ...
To simplify, it's preferable to list these options in text file like this "clang -help > command_clang.txt" and search that it you interest in this text file.

Syntax of "lld-link" command is next :

lld-link /help
OVERVIEW: LLVM Linker

USAGE: lld-link.exe [options] file...

OPTIONS:

- /align: Section alignment
- /aligncomm: Set common symbol alignment
- /allowbind:no Disable DLL binding
- /allowbind Enable DLL binding (default)
- /allowisolation:no Disable DLL isolation
- /allowisolation Enable DLL isolation (default)
- /alternatename: Define weak alias
- /appcontainer:no Image can run outside an app container (default)
- /appcontainer Image can only be run in an app container
- /base: Base address of the program
- /Brepro Use a hash of the executable as the PE header timestamp
- /build-id:no Do not Generate build ID
- /build-id Generate build ID (always on when generating PDB)
- /call-graph-ordering-file:
  - Layout sections to optimize the given callgraph
- /call-graph-profile-sort:no
  - Do not reorder sections with call graph profile
- /call-graph-profile-sort
  - Reorder sections with call graph profile (default)
- /cetcompat:no Don't mark executable image as compatible with Control-flow Enforcement Technology (CET) Shadow Stack (default)
- /cetcompat Mark executable image as compatible with Control-flow Enforcement Technology (CET) Shadow Stack
- --color-diagnostics=[auto,always,never]
  - Use colors in diagnostics (default: auto)

- --color-diagnostics Alias for --color-diagnostics=always
- /debug: Embed a symbol table in the image with option
- /debugtype: Debug Info Options
- /debug Embed a symbol table in the image
- /def: Use module-definition file
- /defaultlib: Add the library to the list of input files
- /delayload: Delay loaded DLL name
- /demangle:no Do not demangle symbols in output
- /demangle Demangle symbols in output (default)
- /dependentloadflag:
  - Sets the default load flags used to resolve the statically linked imports of a module
- /diasdkdir: Set the location of the DIA SDK
- /dll Create a DLL
- /driver:uponly Set IMAGE_DLL_CHARACTERISTICS_WDM_DRIVER bit in PE header
- /driver:wdm Set IMAGE_FILE_UP_SYSTEM_ONLY bit in PE header
- /driver Generate a Windows NT Kernel Mode Driver
- /dwodir: Directory to store .dwo files when LTO and debug fission are used
- /dynamicbase:no Disable ASLR (default when /fixed)
- /dynamicbase Enable ASLR (default unless /fixed)
- /end-lib End group of objects treated as if they were in a library
- /entry: Name of entry point symbol
- /errorlimit: Maximum number of errors to emit before stopping (0 = no limit)
- /exclude-symbols:<symbol[,symbol,...]>
  - Exclude symbols from automatic export
- /export: Export a function
- /filealign: Section alignment in the output file
- /fixed:no Enable base relocations (default)
- /fixed Disable base relocations
- /force:multipleres Allow multiply defined resources when creating executables
- /force:multiple Allow multiply defined symbols when creating executables
- /force:unresolved Allow undefined symbols when creating executables
- /force Allow undefined and multiply defined symbols
- /functionpadmin: Prepares an image for hotpatching
- /guard: Control flow guard
- /heap: Size of the heap
- /highentropyva:no Disable 64-bit ASLR
- /highentropyva Enable 64-bit ASLR (default on 64-bit)

- /ignore: Specify warning codes to ignore
- /implib: Import library name
- /include: Force symbol to be added to symbol table as undefined one
- /includeoptional:
  - Add symbol as undefined, but allow it to remain undefined
- /incremental:no Overwrite import library even if contents are unchanged
- /incremental Keep original import library if contents are unchanged
- /inferasanlibs:no No effect (default)
- /inferasanlibs Unused, generates a warning
- /integritycheck:no No effect (default)
- /integritycheck Set FORCE_INTEGRITY bit in PE header
- /largeaddressaware:no Disable large addresses (default on 32-bit)
- /largeaddressaware Enable large addresses (default on 64-bit)
- /libpath: Additional library search path
- /lib Act like lib.exe; must be first argument if present
- /linkrepro:directory Write repro.tar containing inputs and command to reproduce link
- /lldemit: Specify output type
- /lldignoreenv Ignore environment variables like %LIB%
- /lldltocache: Path to ThinLTO cached object file directory
- /lldltocachepolicy:
  - Pruning policy for the ThinLTO cache
- /lldsavetemps Save intermediate LTO compilation results
- /lto-cs-profile-file:
  - Context sensitive profile file path
- /lto-cs-profile-generate
  - Perform context sensitive PGO instrumentation
- /lto-obj-path: output native object for merged LTO unit to this path
- /lto-pgo-warn-mismatch:no
  - turn off warnings about profile cfg mismatch
- /lto-pgo-warn-mismatch turn on warnings about profile cfg mismatch (default)>
- /machine: Specify target platform
- /manifest: NO disables manifest output; EMBED[,ID=#] embeds manifest as resource in the image
- /manifestdependency:
  - Attributes for element in manifest file; implies /manifest
- /manifestfile: Manifest output path, with /manifest
- /manifestinput: Additional manifest inputs; only valid with /manifest:embed

- /manifestuac: User access control
- /manifest Create .manifest file
- /mapinfo: Include the specified information in a map file
- /merge: Combine sections
- /mllvm: Options to pass to LLVM
- /natvis: Path to natvis file to embed in the PDB
- --no-color-diagnostics Alias for --color-diagnostics=never
- /nodefaultlib: Remove a default library
- /nodefaultlib Remove all default libraries
- /noentry Don't add reference to DllMainCRTStartup; only valid with /dll
- /noimplib Don't output an import lib
- /nxcompat:no Disable data execution provention
- /nxcompat Enable data execution prevention (default)
- /opt: Control optimizations
- /order: Put functions in order
- /out: Path to file to write output
- /pdb: PDB file path
- /pdbaltpath: PDB file path to embed in the image
- /pdbpagesize: PDB page size
- /pdbsourcepath: Base path used to make relative source file path absolute in PDB
- /pdbstream:=
  - Embed the contents of in the PDB as named stream
- /pdbstripped: Stripped PDB file path
- /print-symbol-order:
  - Print a symbol order specified by /call-graph-ordering-file and /call-graph-profile-sort into the specified file
- /release Set the Checksum in the header of an PE file
- /reproduce:filename Write tar file containing inputs and command to reproduce link
- --rsp-quoting= Quoting style for response files, 'windows' (default) or 'posix'
- /safeseh:no Don't produce an image with Safe Exception Handler
- /safeseh Produce an image with Safe Exception Handler (only for x86)
- /section: Specify section attributes
- /stack: Size of the stack
- /start-lib Start group of objects treated as if they were in a library
- /stub: Specify DOS stub file
- /subsystem: Specify subsystem
- /swaprun:cd Make loader run output binary from swap instead of from CD

- /swaprun:net Make loader run output binary from swap instead of from network
- /swaprun: Comma-separated list of 'cd' or 'net'
- /thinlto-emit-imports-files
  - Emit .imports files with -thinlto-index-only
- /thinlto-index-only:
  - -thinlto-index-only and also write native module names to file
- /thinlto-index-only Instead of linking, emit ThinLTO index files
- /thinlto-object-suffix-replace:
  - 'old;new' replace old suffix with new suffix in ThinLTO index
- /thinlto-prefix-replace:
  - 'old;new' replace old prefix with new prefix in ThinLTO outputs
- /threads: Number of threads. '1' disables multi-threading. By default all available hardware threads are used
- --time-trace-granularity=
  - Minimum time granularity (in microseconds) traced by time profiler
- --time-trace= Record time trace to
- --time-trace Record time trace to file next to output
- /timestamp: Specify the PE header timestamp
- /tsaware:no Create non-Terminal Server aware executable
- /tsaware Create Terminal Server aware executable (default)
- /vctoolsdir: Set the location of the VC tools
- /vctoolsversion: Specify which VC tools version to use
- /version: Specify a version number in the PE header
- --version Display the version number and exit
- /vfsoverlay: Path to a vfsoverlay yaml file to optionally look for /defaultlib's in
- /wholearchive: Include all object files from this library
- /wholearchive Include all object files from all libraries
- /winsdkdir: Set the location of the Windows SDK
- /winsdkversion: Specify which SDK version to use
- /winsysroot: Adds several subdirectories to the library search paths
- /WX:no Don't treat warnings as errors (default)
- /WX Treat warnings as errors

You can consult many documentation on site : https://clang.llvm.org/docs/UsersManual.html