Code::Blocks configuration of MSVC compiler tools and Windows SDK

Carsten Arnholm

2016.01.23

Table of Contents

Introduction

Code::Blocks is an open source cross platform IDE for C++ development. It can be used with many different compilers and it runs on Windows, Linux and Mac. The reasons for using Code::Blocks are several. For example, it provides

- Ways of doing cross platform C++ development using a universal IDE
- Ways of targeting different compilers in one and the same project
- Ways of collecting related and dependent projects in workspaces
- Ways of separating project configuration from compiler/toolset configuration
- A useful 'global variable' concept with many practical uses
- The wxSmith RAD tool for wxWidgets GUI development
- and more

This document aims to explain a way to configure Code::Blocks on Windows using different Visual Studio C++ compilers and Windows SDKs. Where relevant it is discussed how to compile for x86 (32bit) and x64 (64bit) executables.

The general approach taken is to employ the Code::Blocks global variables for configuration of the compiler and related tools, i.e. set the global variable values according to the needs of the compiler, SDK and target processor architecture. Examples are provided for MSVC2010 and MSVC2013. By extension, it should be feasible to reconfigure for other MSVC versions.

It should be noted that there isn't a very clear-cut way to specify compilation to x64 (64 bit) using the MSVC command line tools, using only command line options is not enough. This affects how to set up and use Code::Blocks for 64bit compilation with MSVC, so a chapter is set aside discussing this issue.

Motivation

We need to configure the Compiler and Windows SDK with Code::Blocks, but the question is how. To explore the answer, we must first describe the challenges in some detail. Understanding the problem is key to finding a good solution.

There are several combinations of MSVC tools makes it rather confusing. For example, if we look at a small subset:

- Visual Studio 2010 – x86 mainly
  - Visual Studio 2010 Express + Windows 7.0A SDK (free)
  - Visual Studio 2010 Enterprise + Windows 7.0A SDK (non-free)
- Visual Studio 2013 – x86, x64, (arm)
  - Visual Studio 2013 Express for Windows Desktop + Windows Kit 8.1 (free)
  - Visual Studio 2013 Professional + Windows Kit 8.1 (non-free)

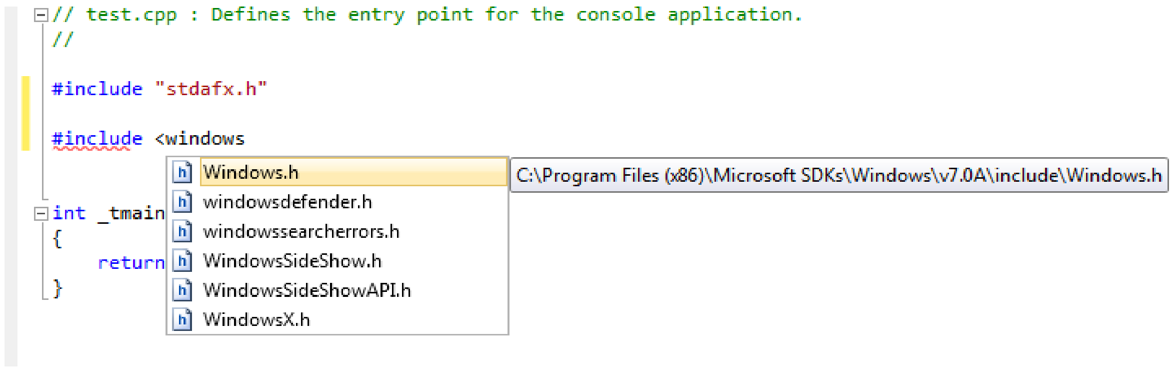
The scenarios for installing SDKs for these combinations are somewhat unique with respect to what toolkits are available, which SDK's to use and where things end up in the filesystem. Even the names of MS IDE is variable . A nice little mess!!

With MSVC2010, the SDK is a separate install. With the MSVC2013 Express/Professional versions you don't need to explicitly install an SDK for ordinary “desktop” development, the SDK is now called a “Windows Kit”. The Windows Kits are found in places like

C:\Program Files (x86)\Windows Kits\8.1

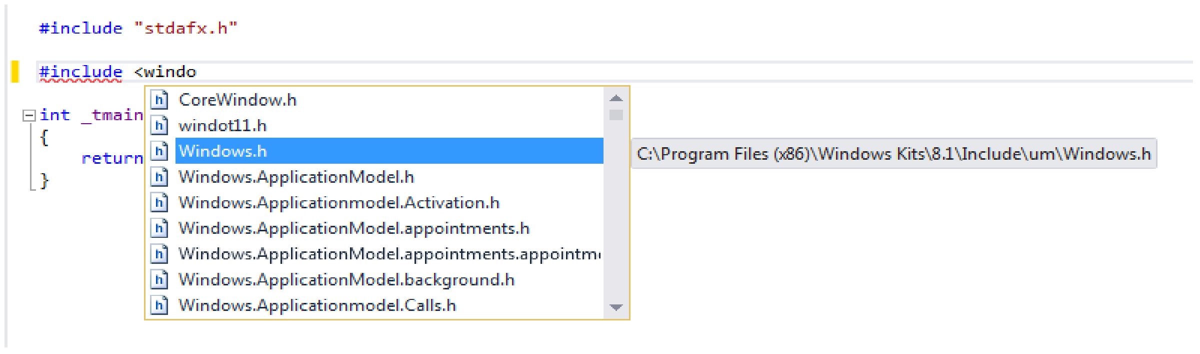
Also, the structure of this “kit” is different from the former SDK, and this must be taken into account when configuring Code::Blocks for use with MSVC.

The location of the 7.0A SDK is confirmed by the the tooltip in Visual Studio 2010 Express IDE:



```
// test.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

#include <windows

int _tmain
{
    return
}
```

The location of the Windows Kit is confirmed by the the tooltip in Visual Studio 2013 Express IDE:



```
#include "stdafx.h"

#include <windo

int _tmain
{
    return
}
```

CoreWindow.h
windot11.h
Windows.h
Windows.ApplicationModel.h
Windows.Applicationmodel.Activation.h
Windows.ApplicationModel.appointments.h
Windows.ApplicationModel.appointments.appointm
Windows.ApplicationModel.background.h
Windows.Applicationmodel.Calls.h
C:\Program Files (x86)\Windows Kits\8.1\Include\um\Windows.h

The above illustrates some of the differences that need to be managed and “configured away”. There are more differences, but the above should be enough to motivate a need for some way to configure Code::Blocks for these variants.

Toolsets

It is important to also understand the existence of so-called 'toolsets', consisting of compiler (cl.exe), linker (link.exe) and other related programs. A toolset typically targets a certain type of processor, e.g. 'x86', 'amd64', 'arm' etc. The implication is that a Visual Studio installation will contain several different variants of cl.exe, link.exe and we must make sure to use the correct ones in combination with the corresponding toolset-dependent compiler and SDK libraries. This adds to the requirements to the configuration system.

A systematic configuration pattern

The seemingly complex and ever-changing state of the compiler and SDK tools for MSVC, needs a configuration pattern to be manageable in Code::Blocks.

We choose a configuration pattern with the following characteristics:

- Always use a "universal compiler" called MSVC, never a specific version. This makes the projects independent of specific MSVC versions. The effect is that the Code::Blocks projects are unaffected by changing MSVC version or target platform.
- Use Code::Blocks global variables to manage the specifics of the compiler/SDK versions. These settings are stored in XML files (*C:\Users\<user>\AppData\Roaming\CodeBlocks\default.conf*) and can therefore be backed up or copied to other users/machines.
- Configure MSVC compiler toolchain and SDK paths via the global variables only

Once set up, the system becomes predictable and reasonably easy to maintain, even when changing MSVC version or targeting a different processor architecture.

Code::Blocks global variables for MSVC

To support the various flavours of MSVC in Code::Blocks, we must manage the MSVC toolchain and the SDK locations separately. We will do this by introducing three key C::B global variables for use with a universal MSVC compiler (called MSVC) in Code::Blocks:

C::B Global Variable	**Description**
MSVC_TOOLCHAIN	Defines MSVC IDE, compiler/linker and related tools
MSVC_SDK	Windows shared SDK corresponding to installed toolchain
MSVC_SDK_ARCH	Windows architecture-specific SDK corresponding to installed toolchain

Example values for some MSVC versions

C::B Global Variable	**MSVC 2010 Express and Enterprise definition**
MSVC_TOOLCHAIN	C:\Program Files (x86)\Microsoft Visual Studio 10.0
MSVC_SDK	C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A
MSVC_SDK_ARCH	Not used, only x86 supported

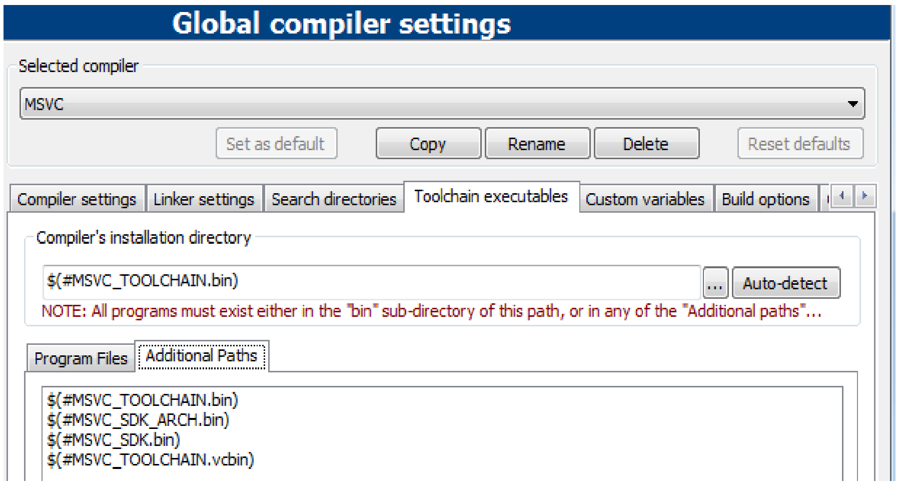
C::B Global Variable	**MSVC 2013 and Professional definition**
MSVC_TOOLCHAIN	C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC
MSVC_SDK	C:\Program Files (x86)\Windows Kits\8.1 (see more below)
MSVC_SDK_ARCH	C:\Program Files (x86)\Windows Kits\8.1 (see more below)

For MSVC2013, more details are explained later in this document.

Universal MSVC compiler and SDK setup

In theory, the following are universal settings regardless of compiler toolchain and SDK version. The actual locations of tools, SDK etc. are defined in global variables. The order of the included paths should be regarded as significant.

Toolchain and paths:



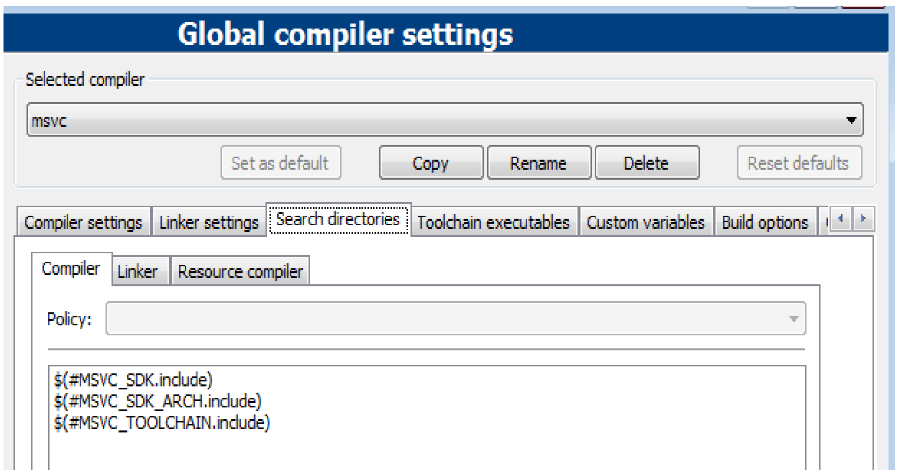
Selected compiler: MSVC

Compiler's installation directory: $(#MSVC_TOOLCHAIN.bin)

NOTE: All programs must exist either in the "bin" sub-directory of this path, or in any of the "Additional paths"...

```
$(#MSVC_TOOLCHAIN.bin)
$(#MSVC_SDK_ARCH.bin)
$(#MSVC_SDK.bin)
$(#MSVC_TOOLCHAIN.vcbin)
```

Compiler include paths:

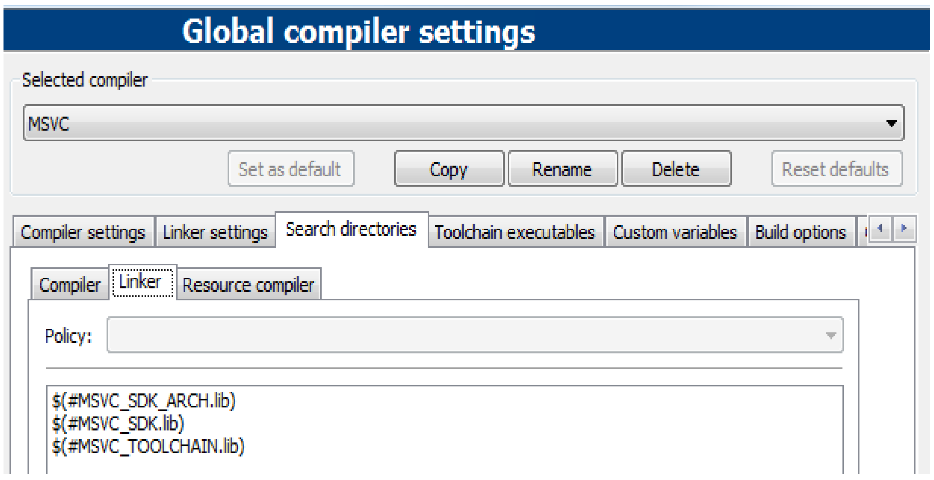


Selected compiler: msvc

Policy:

```
$(#MSVC_SDK.include)
$(#MSVC_SDK_ARCH.include)
$(#MSVC_TOOLCHAIN.include)
```

Linker paths:

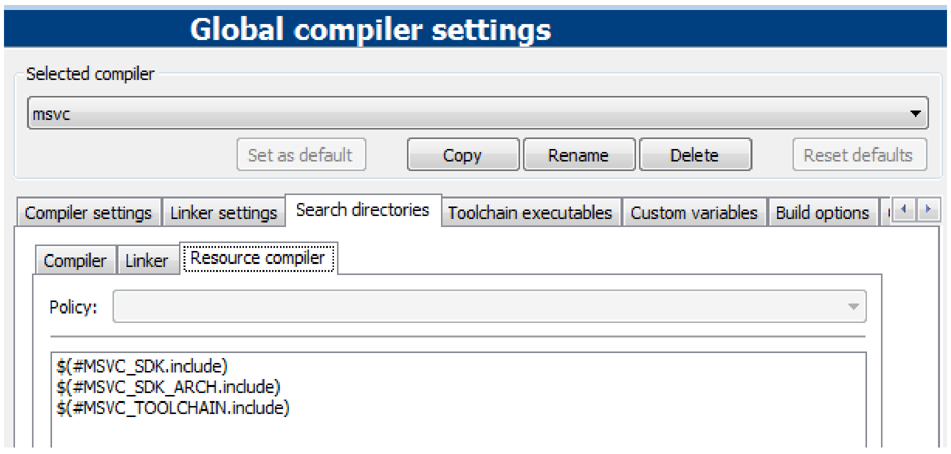


MSVC

Policy:

```
$(#MSVC_SDK_ARCH.lib)
$(#MSVC_SDK.lib)
$(#MSVC_TOOLCHAIN.lib)
```

Resource compiler include paths:



msvc

Policy:

```
$(#MSVC_SDK.include)
$(#MSVC_SDK_ARCH.include)
$(#MSVC_TOOLCHAIN.include)
```

MSVC version & architecture dependent definitions

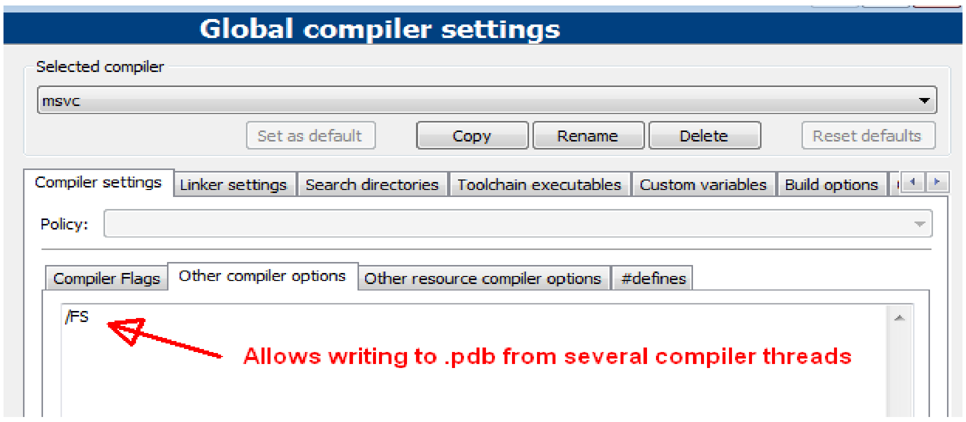
With this we mean

- Which version of Visual Studio is used
- Which version of the Windows SDK/Kit is used
- Which MSVC compiler toolchain variant is ('toolset') used for building
- Which target architecture is chosen (e.g. x86 vs x64)

Generally, the version of Visual Studio installed will dictate which version of the SDK to use, but sometimes the SDK is updated, and we need to tell Code::Blocks which one to use anyway.

With MSVC2013 there are several compiler toolset to choose from, so we need to say which one to use. And finally we need to set the selected target architecture for building our programs. In the following, we list the detailed settings for some relevant configurations.

A half-related feature is the compiler option /FS which allows using many compiler threads in Code::Blocks. This is needed in MSVC2013 but does not exist in MSVC2010.

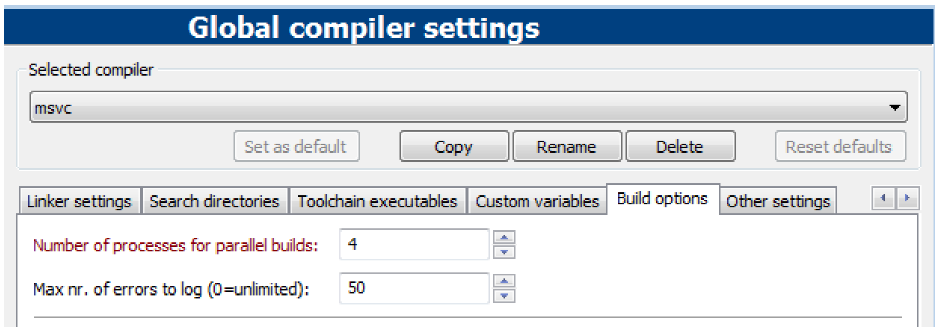


Selected compiler: msvc

/FS

Allows writing to .pdb from several compiler threads

By enabling this option and tell Code::Blocks to use as many processes for parallel build as there are CPU kernels, the compilation can be improved significantly:



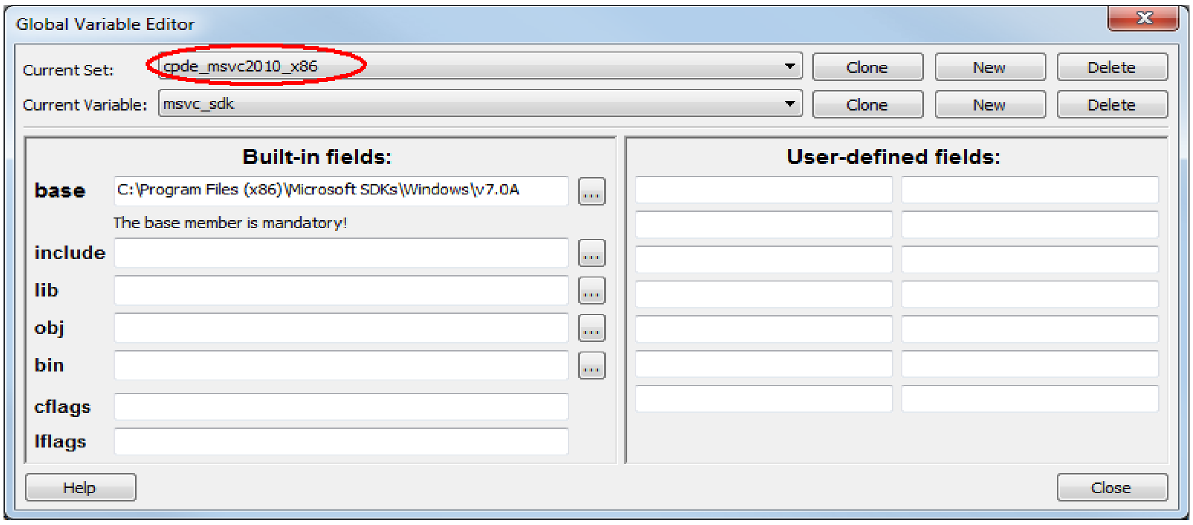
Selected compiler: msvc

Number of processes for parallel builds: 4

Max nr. of errors to log (0=unlimited): 50

cpde_msvc2010_x86

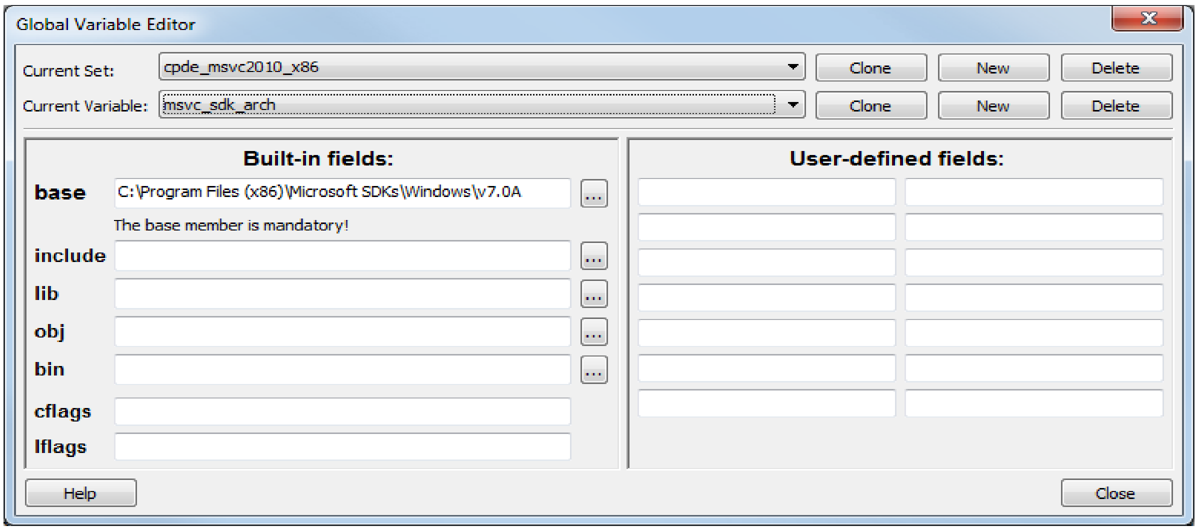
Settings for MS Visual Studio 2010, targeting x86



Current Set: cpde_msvc2010_x86

Current Variable: msvc_sdk

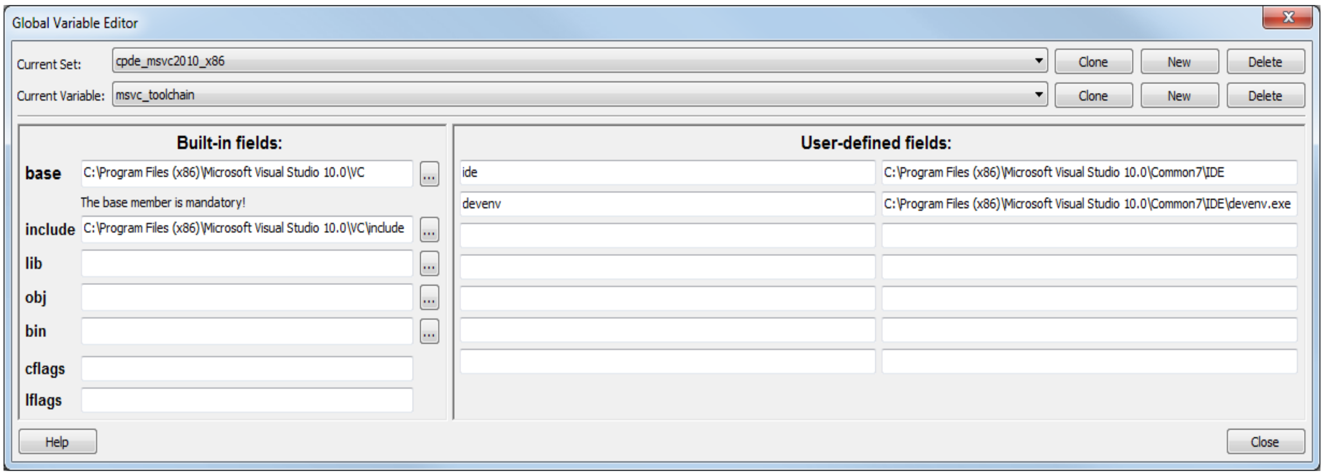
base	C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A
	The base member is mandatory!
include	
lib	
obj	
bin	
cflags	
lflags	



Current Set: cpde_msvc2010_x86

Current Variable: msvc_sdk_arch

base	C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A
	The base member is mandatory!
include	
lib	
obj	
bin	
cflags	
lflags	



Current Set: cpde_msvc2010_x86

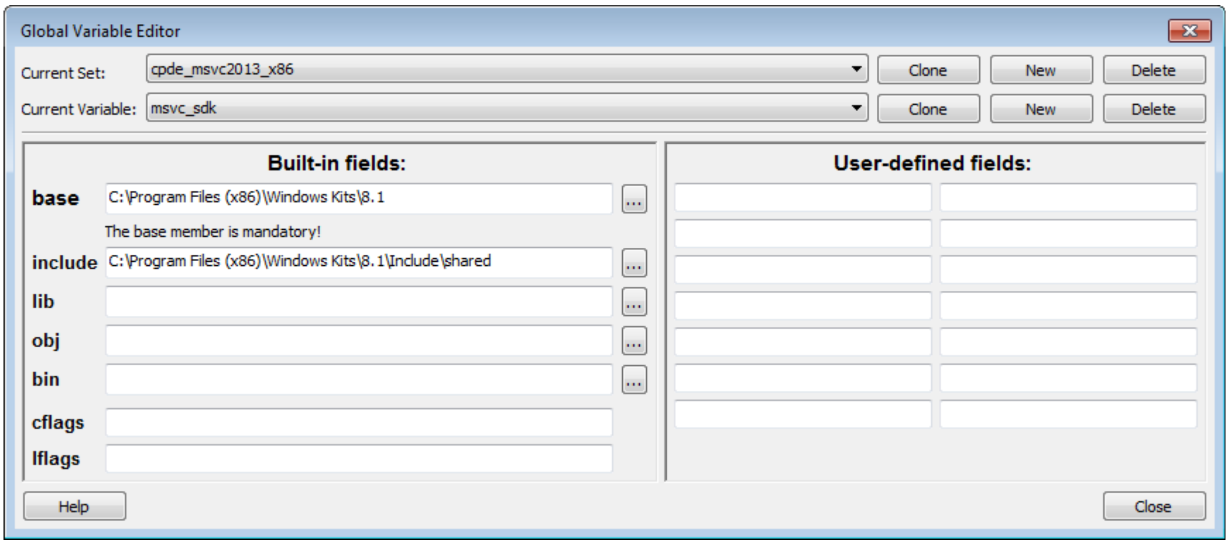
Current Variable: msvc_toolchain

base	C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC
	The base member is mandatory!
include	C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\include
lib	
obj	
bin	
cflags	
lflags	

ide	C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE
devenv	C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\devenv.exe

cpde_msvc2013_x86

Settings for MS Visual Studio 2013, targeting x86 (32bit)

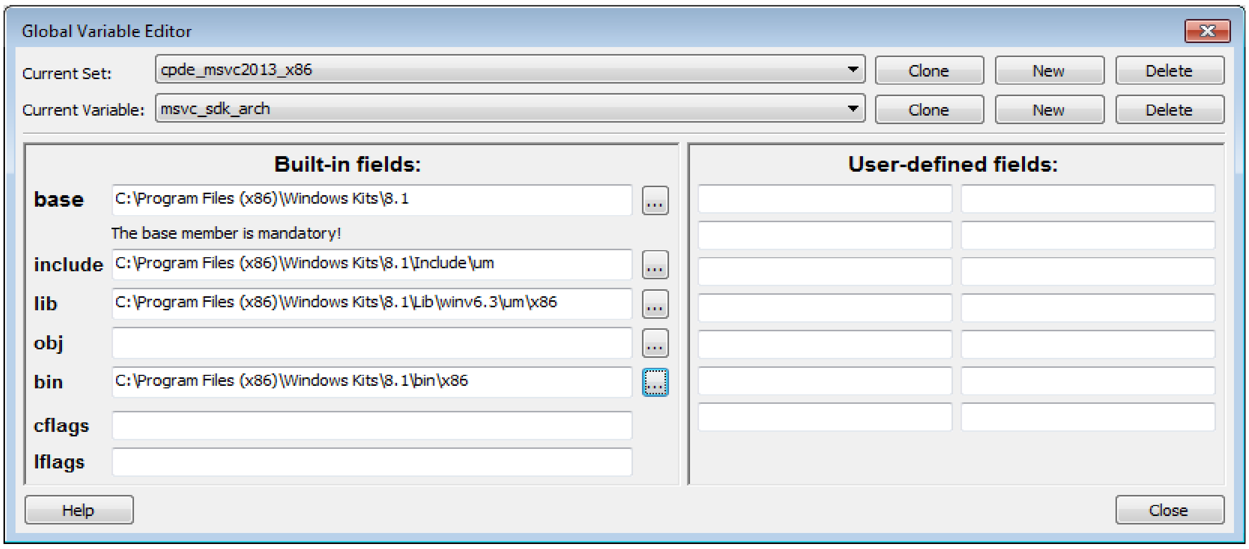


Current Set: cpde_msvc2013_x86
Current Variable: msvc_sdk

Built-in fields:

base	C:\Program Files (x86)\Windows Kits\8.1
	The base member is mandatory!
include	C:\Program Files (x86)\Windows Kits\8.1\Include\shared
lib	
obj	
bin	
cflags	
lflags	

User-defined fields:

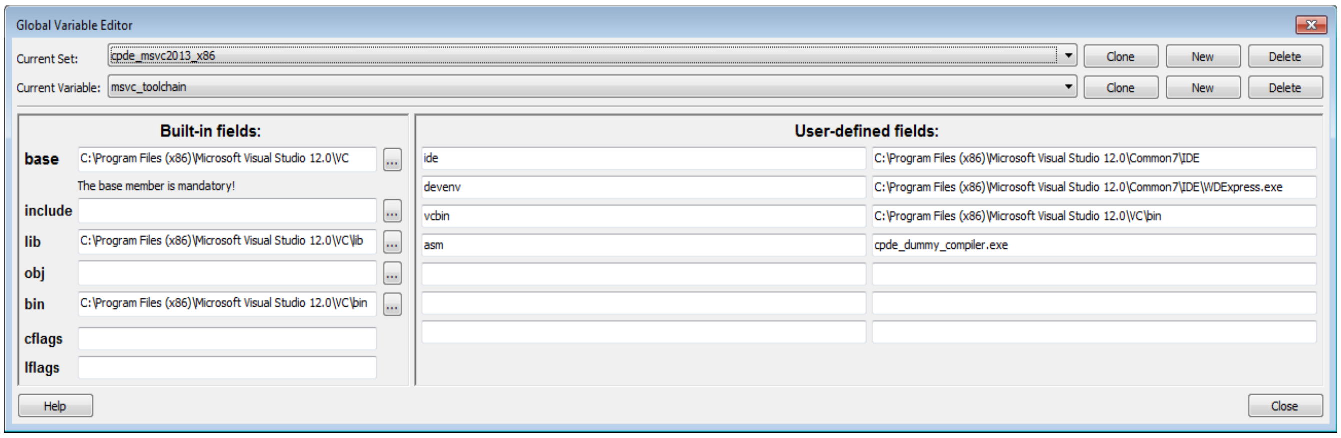


Current Set: cpde_msvc2013_x86
Current Variable: msvc_sdk_arch

Built-in fields:

base	C:\Program Files (x86)\Windows Kits\8.1
	The base member is mandatory!
include	C:\Program Files (x86)\Windows Kits\8.1\Include\um
lib	C:\Program Files (x86)\Windows Kits\8.1\Lib\winv6.3\um\x86
obj	
bin	C:\Program Files (x86)\Windows Kits\8.1\bin\x86
cflags	
lflags	

User-defined fields:



Current Set: cpde_msvc2013_x86
Current Variable: msvc_toolchain

Built-in fields:

base	C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC
	The base member is mandatory!
include	
lib	C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\lib
obj	
bin	C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin
cflags	
lflags	

User-defined fields:

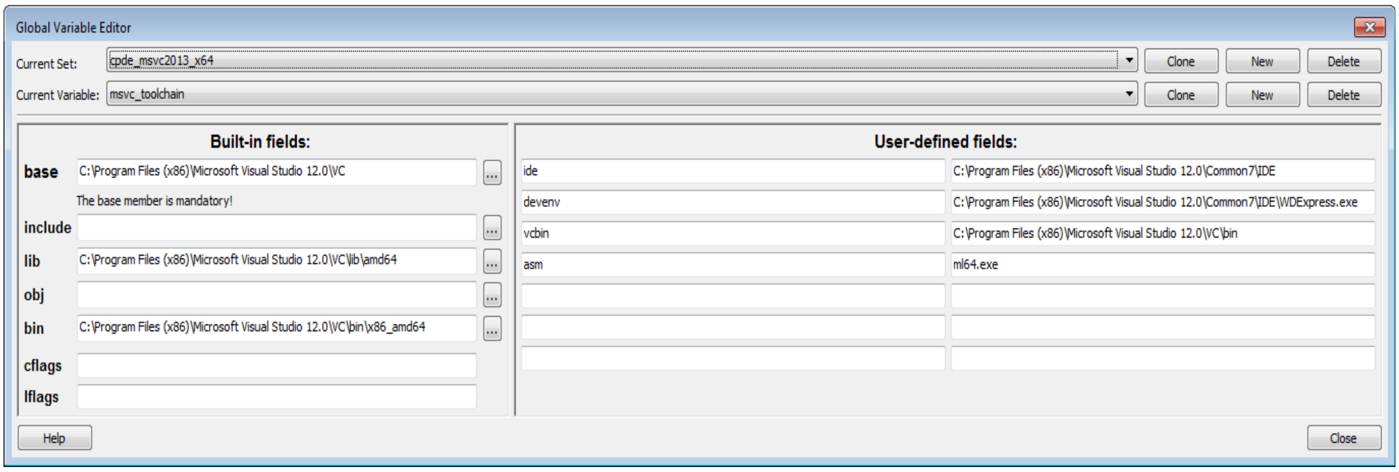
ide	C:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE
devenv	C:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE\WDExpress.exe
vcbin	C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin
asm	cpde_dummy_compiler.exe

(Above shown for for MSVC2013 Express, for non-express change WDexpress.exe to devenv.exe)

cpde_msvc2013_x64

Settings for MS Visual Studio 2013, targeting x64 (64 bit) using x64 native compiler.

The bin directory ends with 'amd64' in the example below, indicating the native compiler toolset. For the free Express version of MSVC2013, the native toolset 'amd64' may not be available. You can then use the 'x86_amd64' cross compiler toolset instead.



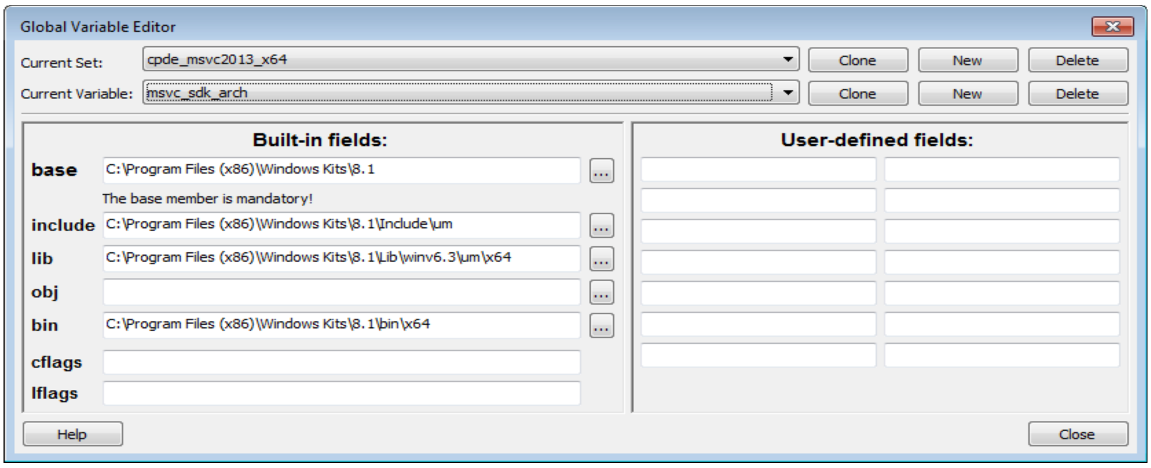
Current Set: cpde_msvc2013_x64
Current Variable: msvc_toolchain

Built-in fields:

base	C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC
	The base member is mandatory!
include	
lib	C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\lib\amd64
obj	
bin	C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\x86_amd64
cflags	
lflags	

User-defined fields:

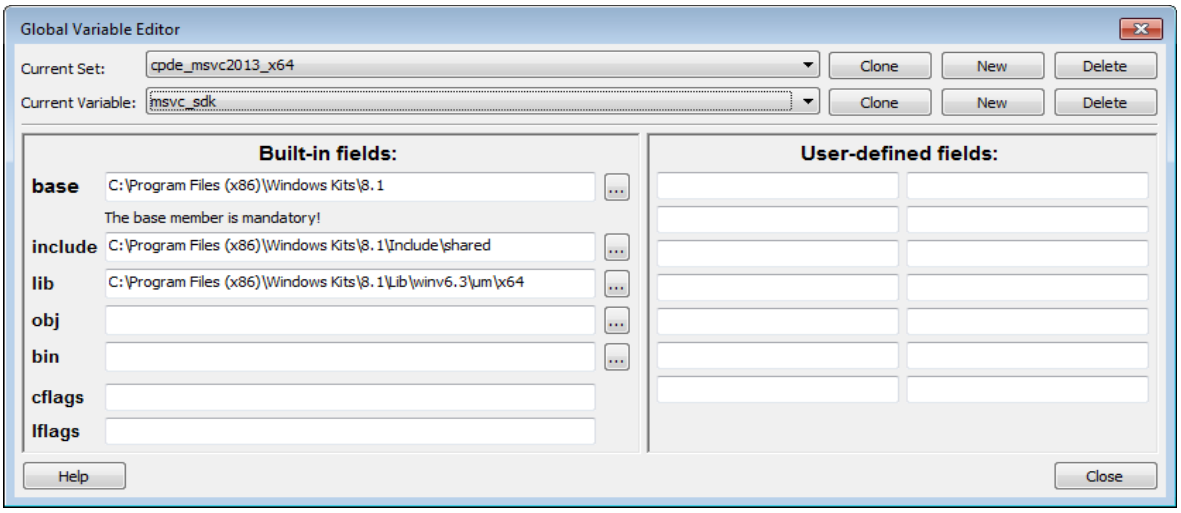
ide	C:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE
devenv	C:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE\WDExpress.exe
vcbin	C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin
asm	ml64.exe



Current Set: cpde_msvc2013_x64
Current Variable: msvc_sdk_arch

Built-in fields:

base	C:\Program Files (x86)\Windows Kits\8.1
	The base member is mandatory!
include	C:\Program Files (x86)\Windows Kits\8.1\Include\um
lib	C:\Program Files (x86)\Windows Kits\8.1\Lib\winv6.3\um\x64
obj	
bin	C:\Program Files (x86)\Windows Kits\8.1\bin\x64
cflags	
lflags	



Current Set: cpde_msvc2013_x64
Current Variable: msvc_sdk

Built-in fields:

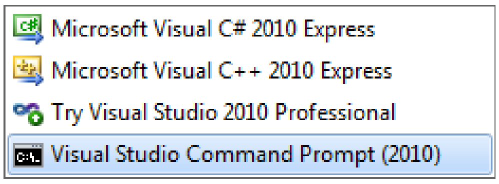
base	C:\Program Files (x86)\Windows Kits\8.1
	The base member is mandatory!
include	C:\Program Files (x86)\Windows Kits\8.1\Include\shared
lib	C:\Program Files (x86)\Windows Kits\8.1\Lib\winv6.3\um\x64
obj	
bin	
cflags	
lflags	

Building libraries using selected MSVC toolchain

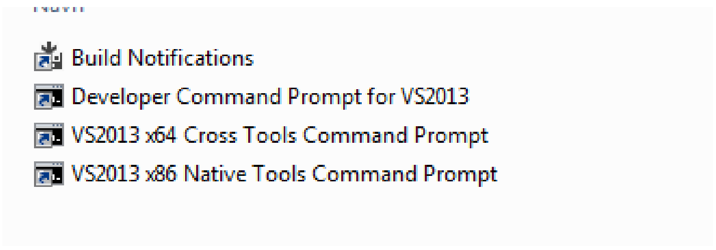
When targeting a specific platform, we must also build the libraries targeting the same platform. Usually, this means we use the same toolchain when building the libraries as we use within Code::Blocks for our own application development.

One way to select compiler toolchain when building from command line is to use the relevant “Command Prompt” for the installed compiler, accessible from the start menu

MSVC2010 Express:


Microsoft Visual C# 2010 Express
Microsoft Visual C++ 2010 Express
Try Visual Studio 2010 Professional
Visual Studio Command Prompt (2010)

MSVC2013 Express:


Build Notifications
Developer Command Prompt for VS2013
VS2013 x64 Cross Tools Command Prompt
VS2013 x86 Native Tools Command Prompt

However, this is often not sufficient. A more predictable way to compile libraries such as boost and wxWidgets is to do it from a .bat file and calling the MSVC procedure for setting up the toolchain environment and setting the correct parameters according to the build system in use. This makes things easier, as we don't need to use the cumbersome and specialised “Command prompts” above, because environment variables pointing tp the MSVC installation are always available

For example, if MSVC2010 (a.k.a 'VS100') and MSVC2013 (a.k.a 'VS120') are installed, the following environment variables will be available:

```
VS100COMNTOOLS=C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\Tools\
VS120COMNTOOLS=C:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\Tools\
```

We can then exploit this in calls within .bat files to select the required toolchain:

```
call "%VS120COMNTOOLS%..\..\VC\vcvarsall.bat" x86_amd64
```

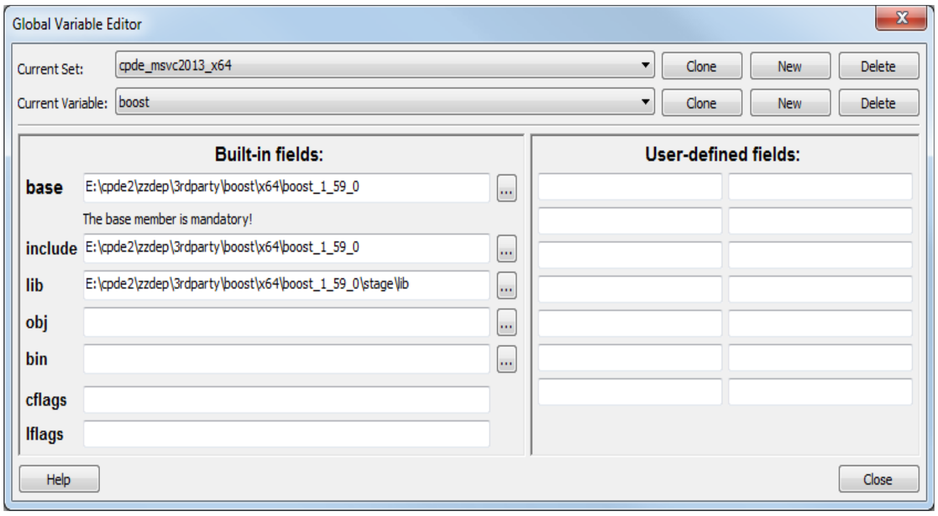
Thie example selects MSVC2013 (Express) cross compilation from x86 targeting x64

Building boost 64bit

The example is for MSVC2013 (Express) cross compilation from x86 targeting x64 of boost 1.59.0

```
REM By using 'call' we can call other batch files,
REM without aborting the execution of the calling batch file,
REM and using the same environment for both batch files.

REM 32bit builds
REM For MSVC2013 Express/Professional: x86

REM 64bit builds
REM MSVC2013 Express          :  x86_amd64  (cross-compiler)
REM For MSVC2013 Professional:  amd64      (native compiler)
REM
call "%VS120COMNTOOLS%..\..\VC\vcvarsall.bat" x86_amd64

REM We assume this batch procedure is placed one level above the boost package,
REM So we change the directory to bootstrap and build the boost libraries.
REM Bootstrap will generate the b2 build system, .\b2 will do the actual build

cd boost_1_59_0

REM Before running bootstrap, remember to edit the line
REM from "set toolset=msvc"
REM to   "set toolset=msvc : 120 ;"

call bootstrap.bat

REM Set address-model=32 for x86 32-bit builds
REM Set address-model=64 for x86_amd64 cross compiler or native amd64 build
REM Set architecture=x86 in all cases

.\b2 -j8 toolset=msvc-12.0 address-model=64 architecture=x86 ^
     link=static threading=multi runtime-link=shared –build-type=minimal

echo "==== boost_1_59_0 build completed ====="
```

A corresponding C::B global variable definition for boost:



Global Variable Editor

Current Set: cpde_msvc2013_x64

Current Variable: boost

Built-in fields:

base	E:\cpde2\zzdep\3rdparty\boost\x64\boost_1_59_0
	The base member is mandatory!
include	E:\cpde2\zzdep\3rdparty\boost\x64\boost_1_59_0
lib	E:\cpde2\zzdep\3rdparty\boost\x64\boost_1_59_0\stage\lib
obj	
bin	
cflags	
lflags	

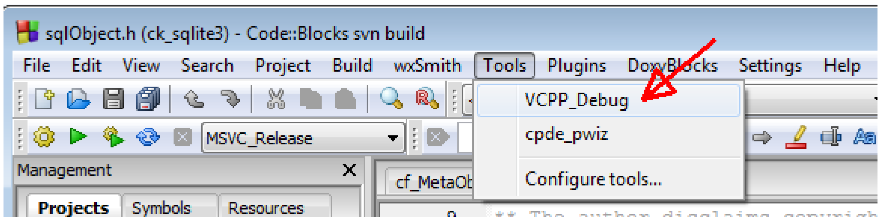
Building wxWidgets 64bit

The example is for MSVC2013 (Express) cross compilation from x86 targeting x64 of wxWidgets 3.0.2

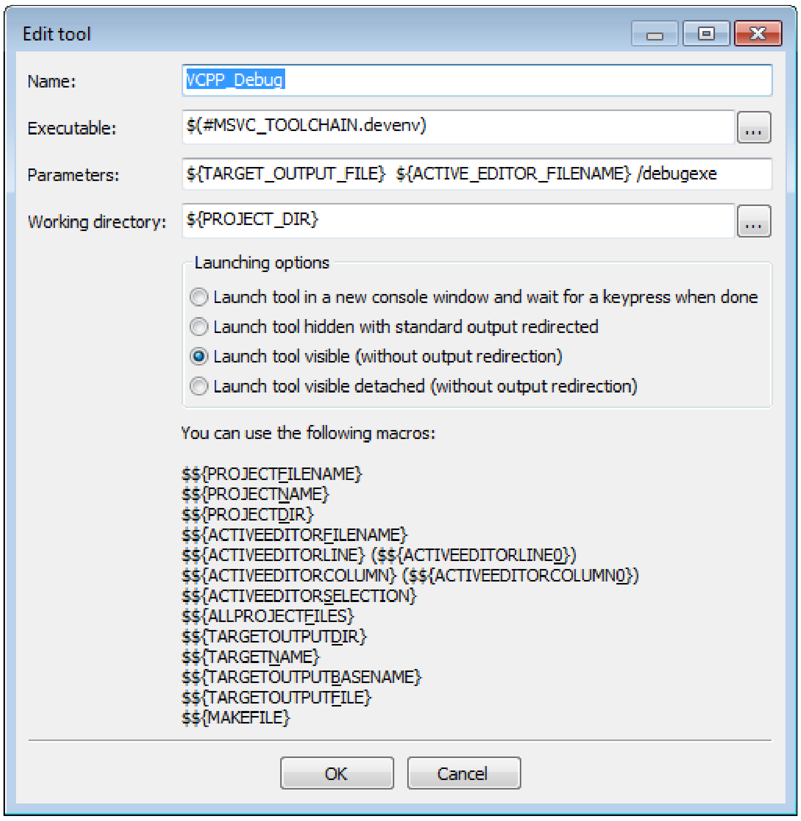
```
REM  This procedure builds wxWidgets on Windows using MSVC and NMAKE
REM  Run from "Visual Studio Command Prompt"
REM   Visual Studio tools -> VS2013 x64 Cross Tools Command Prompt
REM
REM  change dir to build\msw folder, for example
REM     ...\wx\x64\3.0.2\lib\build\msw
REM
REM make sure foreign makeflags don't interfere (QNX does this)
set MAKEFLAGS=

REM Call Visual Studio 2013 to set up the 64bit cross-compiler toolkit.
REM (MSVC2013 Express does not contain the native 'amd64' toolkit)
call "%VS120COMNTOOLS%..\..\VC\vcvarsall.bat" x86_amd64

REM Build release libraries
nmake -f makefile.vc TARGET_CPU=amd64 BUILD=release DEBUG_INFO=1 ^
              RUNTIME_LIBS=dynamic SHARED=0

REM Build debug libraries
nmake -f makefile.vc TARGET_CPU=amd64 BUILD=debug   DEBUG_INFO=1 ^
              RUNTIME_LIBS=dynamic SHARED=0

echo "==== wxWidgets 3.0.2 build completed ====="
```

Setting up Tools menu for debugging

A possibility is to set up the CDB.exe gdb-like debugger for MSVC in Code::Blocks, but this is really a separate research issue. Instead we rely on starting the MSVC IDE for debugging an open project, using the C::B Tools menu:


sqlObject.h (ck_sqlite3) - Code::Blocks svn build
File Edit View Search Project Build wxSmith Tools Plugins DoxyBlocks Settings Help
MSVC_Release
Management
Projects Symbols Resources
cf_MetaOb
VCPP_Debug
cpde_pwiz
Configure tools...

defined as below for MSVC2013 Express (the non-free MSVC version requires a slightly different setup for the "Parameters" line):


Edit tool

Name:	VCPP_Debug
Executable:	$(#MSVC_TOOLCHAIN.devenv)
Parameters:	${TARGET_OUTPUT_FILE} ${ACTIVE_EDITOR_FILENAME} /debugexe
Working directory:	${PROJECT_DIR}

Launching options

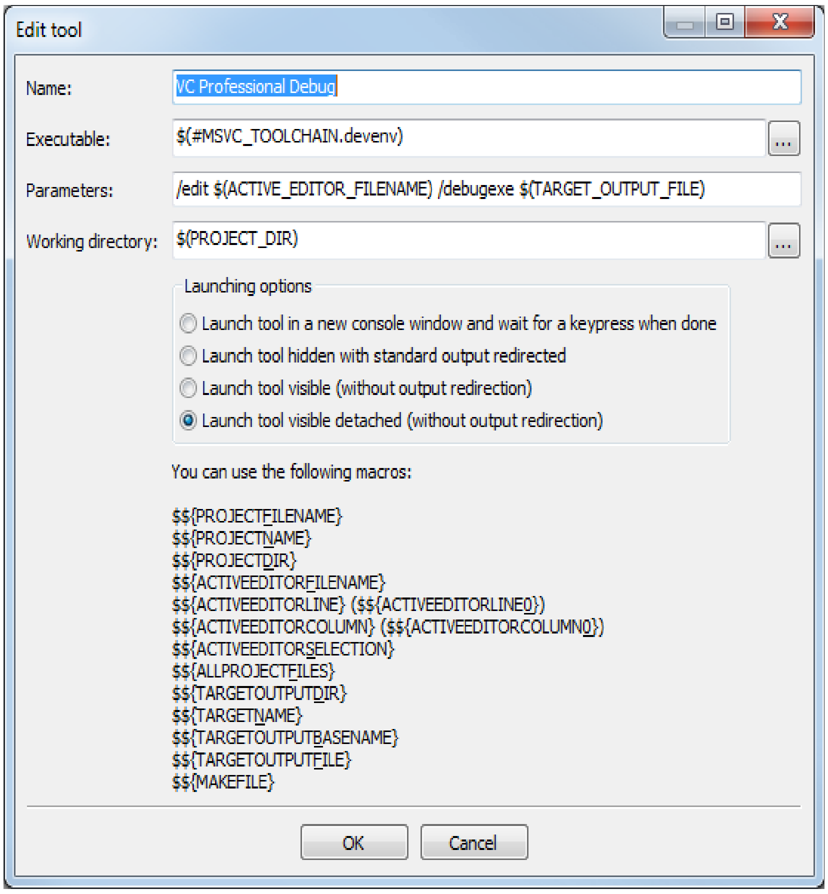
- Launch tool in a new console window and wait for a keypress when done
- Launch tool hidden with standard output redirected
- [✓ Launch tool visible (without output redirection)]
- Launch tool visible detached (without output redirection)

You can use the following macros:

```
$${PROJECTFILENAME}
$${PROJECTNAME}
$${PROJECTDIR}
$${ACTIVEEDITORFILENAME}
$${ACTIVEEDITORLINE} ($${ACTIVEEDITORLINE0})
$${ACTIVEEDITORCOLUMN} ($${ACTIVEEDITORCOLUMN0})
$${ACTIVEEDITORSELECTION}
$${ALLPROJECTFILES}
$${TARGETOUTPUTDIR}
$${TARGETNAME}
$${TARGETOUTPUTBASENAME}
$${TARGETOUTPUTFILE}
$${MAKEFILE}
```

When we build a Debug target in Code::Blocks, we can open the file we want to debug, then use the Tools menu to trigger the MSVC IDE/debugger with all the usual debugger features. While an integrated debugger is more elegant, the MSVC debugger is quite good, so this simple setup works quite well in practice.

The same thing done using VC Professional (next page). Notice the parameters are different.


Edit tool

Name: VC Professional Debug

Executable: $(#MSVC_TOOLCHAIN.devenv)

Parameters: /edit $(ACTIVE_EDITOR_FILENAME) /debugexe $(TARGET_OUTPUT_FILE)

Working directory: $(PROJECT_DIR)

Launching options

- Launch tool in a new console window and wait for a keypress when done
- Launch tool hidden with standard output redirected
- Launch tool visible (without output redirection)
- [✓ Launch tool visible detached (without output redirection)]

You can use the following macros:

```
$${PROJECTFILENAME}
$${PROJECTNAME}
$${PROJECTDIR}
$${ACTIVEEDITORFILENAME}
$${ACTIVEEDITORLINE} ($${ACTIVEEDITORLINE0})
$${ACTIVEEDITORCOLUMN} ($${ACTIVEEDITORCOLUMN0})
$${ACTIVEEDITORSELECTION}
$${ALLPROJECTFILES}
$${TARGETOUTPUTDIR}
$${TARGETNAME}
$${TARGETOUTPUTBASENAME}
$${TARGETOUTPUTFILE}
$${MAKEFILE}
```

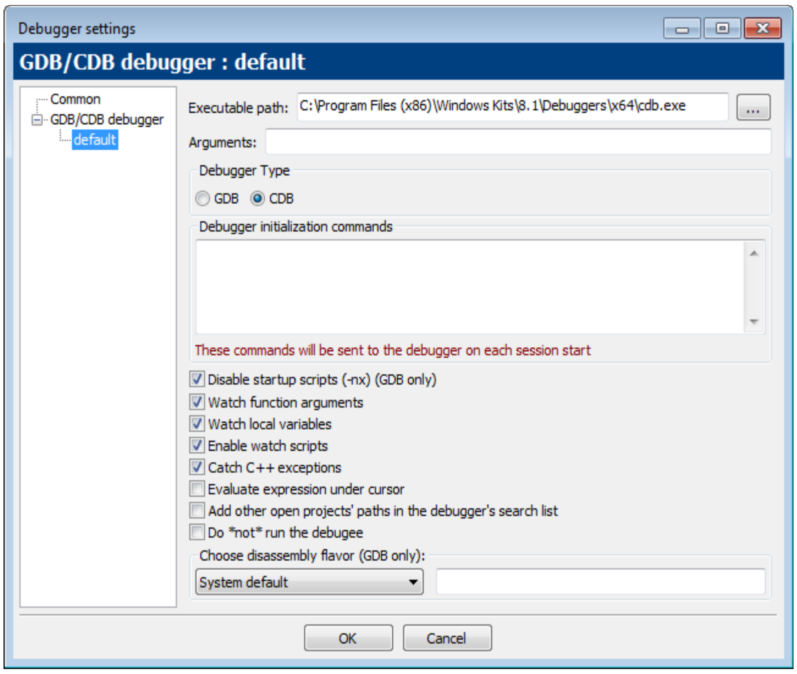
Using CDB.exe for debugging within Code::Blocks

If you don't like the Tools menu approach using the Visual Studio IDE as external debugger, there is also a "proper way", using the x86 and the x64 dbg-like debuggers:

```
C:\Program Files (x86)\Windows Kits\8.1\Debuggers\x64\cdb.exe
C:\Program Files (x86)\Windows Kits\8.1\Debuggers\x86\cdb.exe
```

As shown.

The MSVC debugger is however more complete.


Debugger settings

GDB/CDB debugger : default

Executable path: C:\Program Files (x86)\Windows Kits\8.1\Debuggers\x64\cdb.exe

Arguments:

Debugger Type: GDB [✓ CDB]

Debugger initialization commands

These commands will be sent to the debugger on each session start

- [✓ Disable startup scripts (-nx) (GDB only)]
- [✓ Watch function arguments]
- [✓ Watch local variables]
- [✓ Enable watch scripts]
- [✓ Catch C++ exceptions]
- Evaluate expression under cursor
- Add other open projects' paths in the debugger's search list
- Do *not* run the debugee

Choose disassembly flavor (GDB only): System default

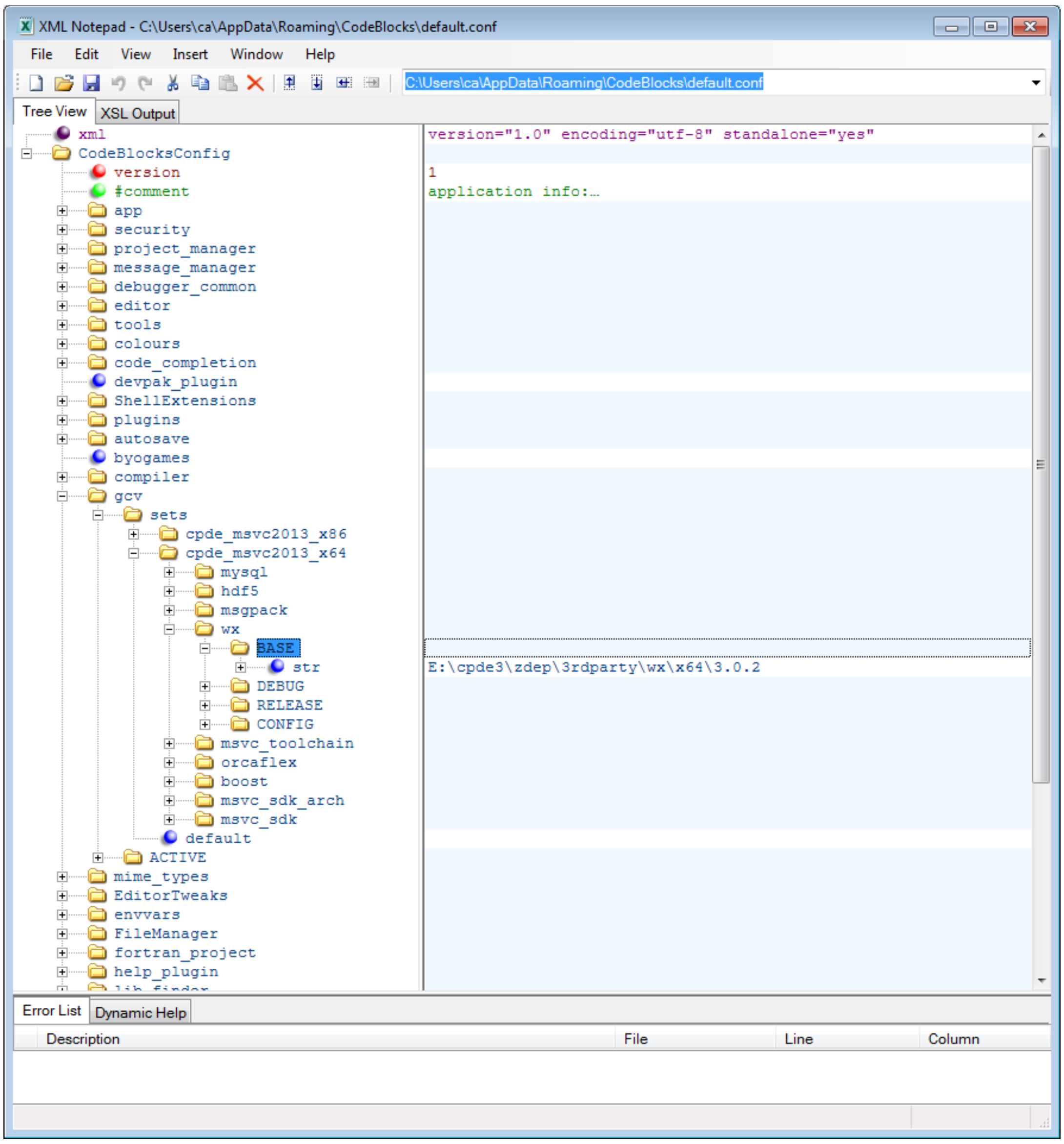
Other useful ideas

It is possible to use XML Notepad 2007 or equivalent to edit the configuration fil at

C:\Users\<user>\AppData\Roaming\CodeBlocks\default.conf

as it really is an XML file. Be very careful doing this as you can easily corrupt your installation.

The global variable definitions are found under <gcv>



XML Notepad - C:\Users\ca\AppData\Roaming\CodeBlocks\default.conf

```
xml                          version="1.0" encoding="utf-8" standalone="yes"
CodeBlocksConfig
  version                    1
  #comment                   application info:…
  app
  security
  project_manager
  message_manager
  debugger_common
  editor
  tools
  colours
  code_completion
  devpak_plugin
  ShellExtensions
  plugins
  autosave
  byogames
  compiler
  gcv
    sets
      cpde_msvc2013_x86
      cpde_msvc2013_x64
        mysql
        hdf5
        msgpack
        wx
          BASE
            str              E:\cpde3\zdep\3rdparty\wx\x64\3.0.2
          DEBUG
          RELEASE
          CONFIG
        msvc_toolchain
        orcaflex
        boost
        msvc_sdk_arch
        msvc_sdk
      default
    ACTIVE
  mime_types
  EditorTweaks
  envvars
  FileManager
  fortran_project
  help_plugin
```