

GPH SDK

Version 10.08

10.02	GPH_SDK for WIZ	2010.01.25
10.08	GPH_SDK for CAANOO	2010.08.09
10.08	tools/target/3g_sensor added.	2010.08.09
10.08	tools/target/haptic_motor added.	2010.08.09
10.08	tools/target/opengles added.	2010.08.09
10.08	DGE/examples added.	2010.09.10

Table of Contents

Chapter 1 Introduction

- 1.1. What is GPH SDK ?
- 1.2. What is DGE ?

Chapter 2 Installing SDK

- 2.1. Installing CodeBlocks
- 2.2. Installing GPH SDK

Chapter 3 Compiler Settings

- 3.1. Native Compiler Settings
- 3.2. Cross Compiler Settings

Chapter 4 New Project

- 4.1. Creating Project
- 4.2. Using Templates

Chapter 5 Project/target options

- 5.1. Project settings
- 5.2. Build targets options
- 5.3. Project build options
- 5.4. Debugger

Chapter 6 Using Makefile

- 6.1. Host "Make" commands
- 6.2. Target "Make" commands
- 6.3. Makefile : example

1. Introduction

This document guides how to install GPH SDK into your workstation, and explains settings that are needed to use GPH SDK.

1.1. What is GPH SDK ?

GPH SDK is a software development tool kit dedicated for developing software for the handheld device developed by Game Park Holdings. Via GPH SDK, Developers can develop applications or game software. GPH SDK mainly consists of the following 2 features.

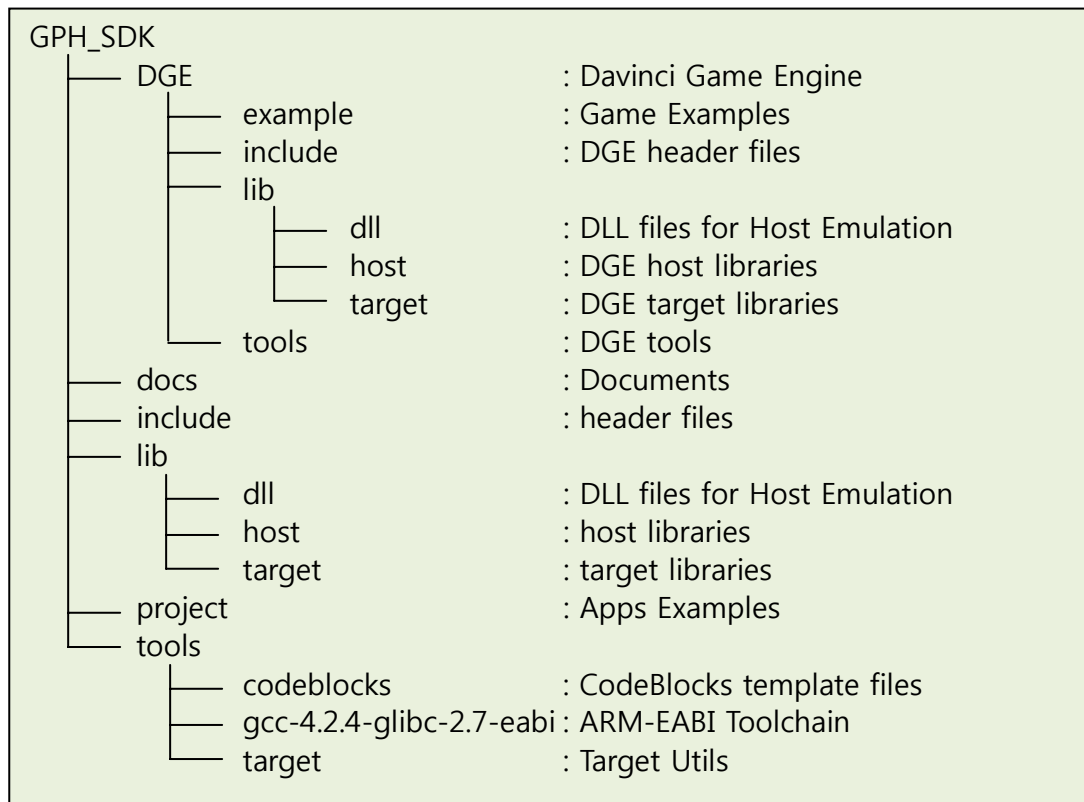
- Game Engine DGE (Davinci Game Engine)
- Various libraries and header files

GPH SDK supports various IDE(Integrated Development Environment). This document showcases GPH SDK using CodeBlocks as an example. For Visual Studio or Visual C++ users, please refer to other documents. GPH SDK is available in two different distribution versions.

- Distribution version for Window user
- Distribution version for Linux user

With simultaneous support of host compiling and cross compiling, GPH SDK allows developers to conveniently create game software or applications. It also supports a great debugging solution that developer may host-emulate and cross-debug at the same time.

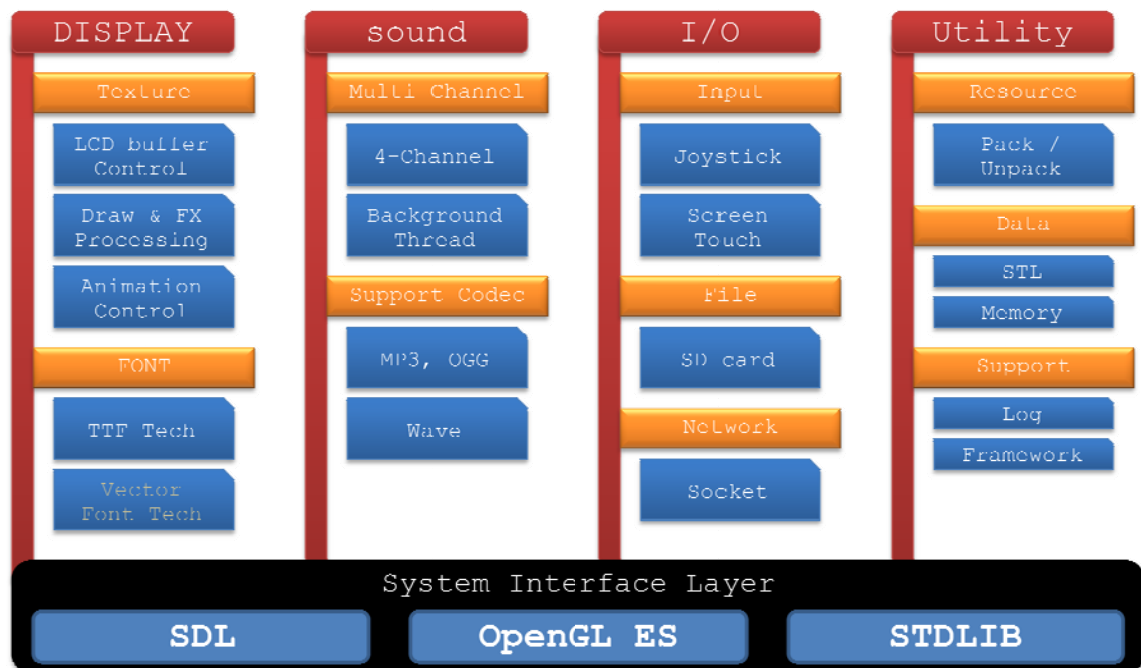
The directory of GPH SDK is structured as following diagram.



<Fig 1.1> Directory Structure

1.2. What is DGE ?

DGE is an abbreviated term of "Davinci Game Engine" and is specifically designed to develop software that runs on the game devices manufactured by GamePark Holdings. DGE is mainly based on SDL and OpenGL to offer a comfortable developing environment. The generic structure of DGE is as below.



<Fig 1.2> Davinci Architecture

2. Installing SDK

2.1. Installing CodeBlocks

CodeBlocks is an Integrated Development Environment that supports open source and multi-platforms. Click on the listed URL to download necessary installation files that suit your project, either Window or Linux etc.

<http://www.codeblocks.org>

1) For Window OS

If you are using Window OS, you should use MinGW contained in CodeBlocks as a cross compiler. So please download and Install CodeBlocks which contains MinGW.

2) For Linux OS

```
[Ubuntu 9.04]
# sudo apt-get install codeblocks
# sudo apt-get install g++
# sudo apt-get install libgl1-mesa-glx
[Fedora 10]
# yum install codeblocks
```

2.2. Installing GPH SDK

1) For Window OS

- Run GPH_SDK.exe to start installing GPH SDK
- This document is based on an assumption that GPH SDK is installed at C:\Program Files\CAANOOW\GPH_SDK directory.

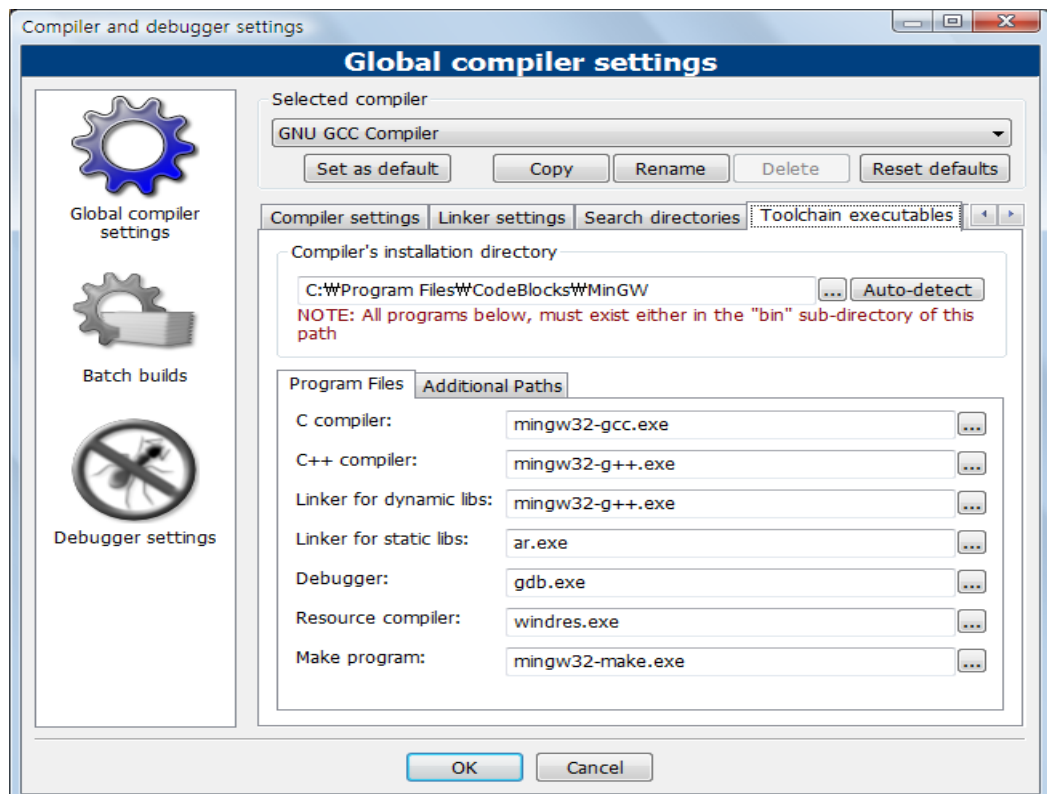
2) For Linux OS

- Unzip the installation file, GPH_SDK.tar.gz, at the home directory.

3. Compiler Settings

3.1. Native Compiler Settings

3.1.1. Compiler installation directory

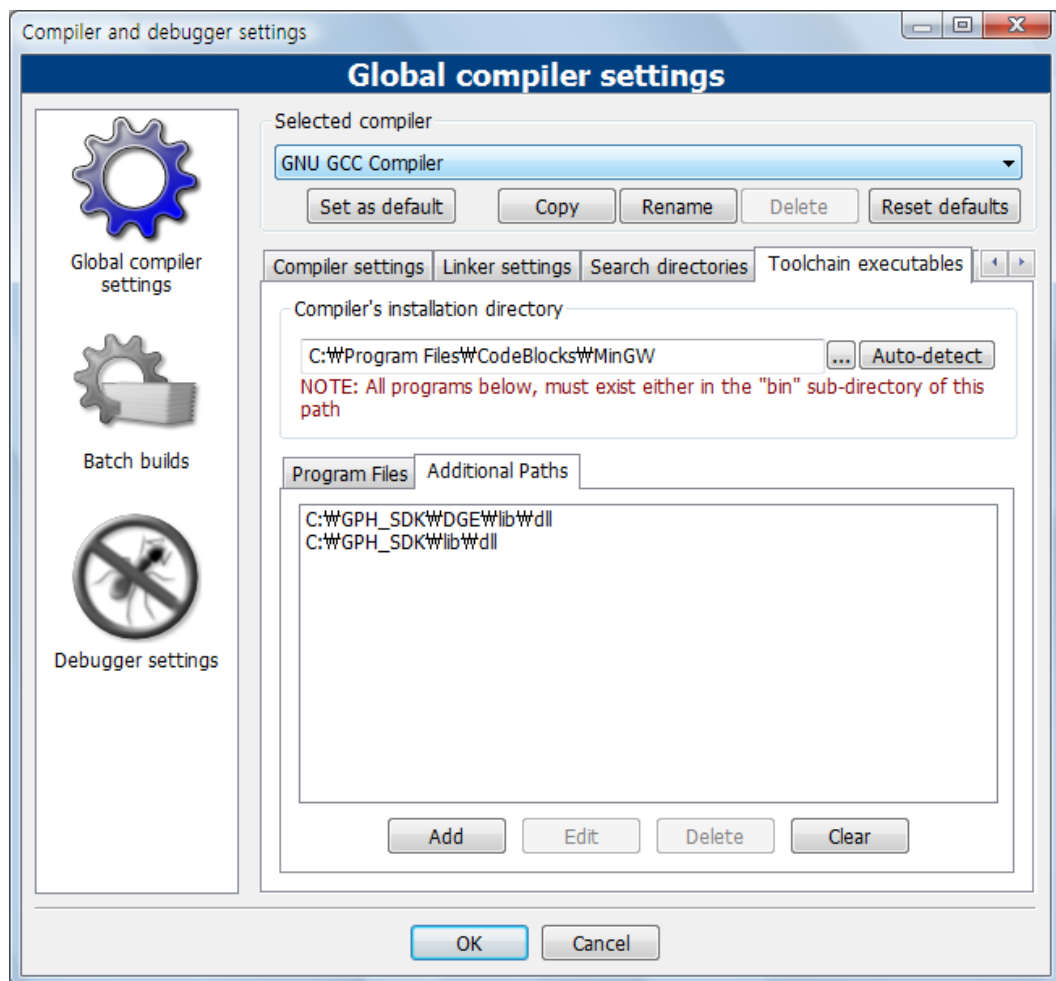


The above screenshot shows settings of host compiler. Host Compiler defaults to MinGW Compiler that gets automatically installed when installing CodeBlocks. It's recommended to use the original setting as shown above. When CodeBlocks is installed in other directory, you can change "Compiler's installation directory" with the directory.

In order to change settings:

1. From "Settings" menu, select "Compiler and Debugger".
2. On "Selected compiler" list, select "GNU GCC Compiler".
3. From tab menu, select "Toolchain executables".
4. From sub-tab menu, select "Program Files".

3.1.2. Additional Paths

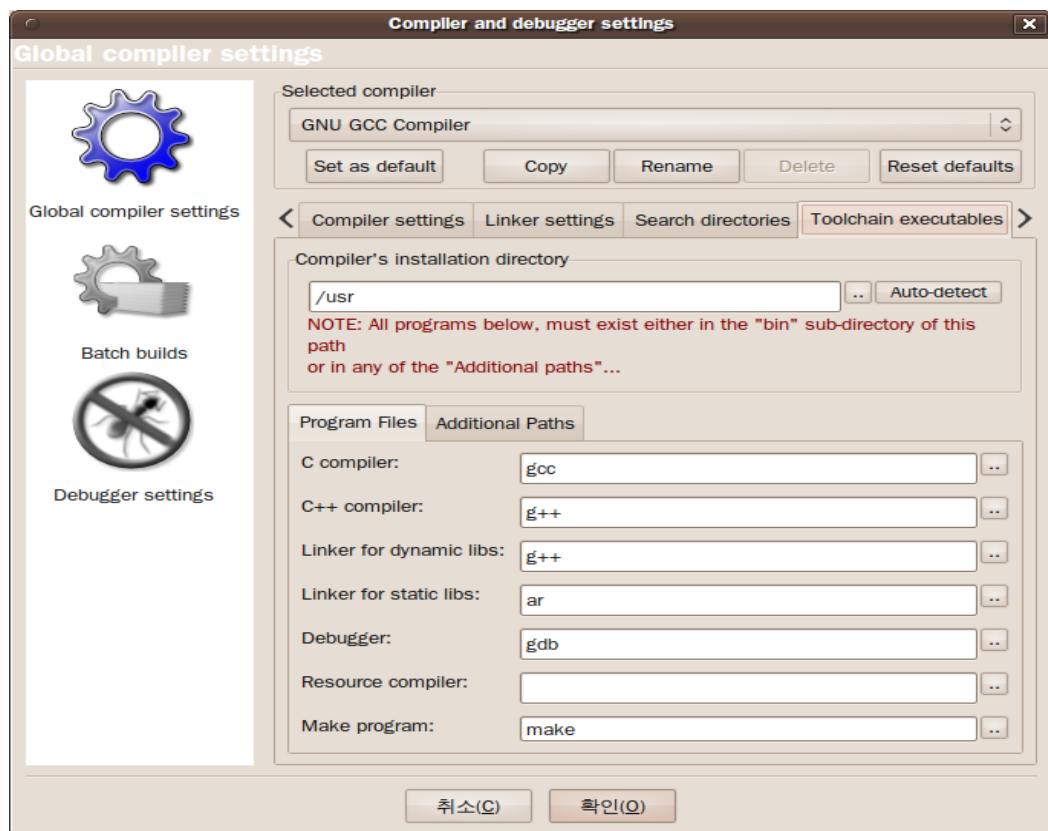


"Additional Paths" assigns an extra path that is needed when compiling or executing a program. The above case shows how the paths are assigned for the directories containing Window DLL files required to compile or execute a program.

In order to change settings:

1. From "Settings" menu, select "Compiler and Debugger".
2. On "Selected compiler" list, select "GNU GCC Compiler".
3. From tab menu, select "Toolchain executables".
4. From sub-tab menu, select "Additional Paths".
5. Click "Add" button to add a directory.

3.1.3. Compiler installation directory (Linux)



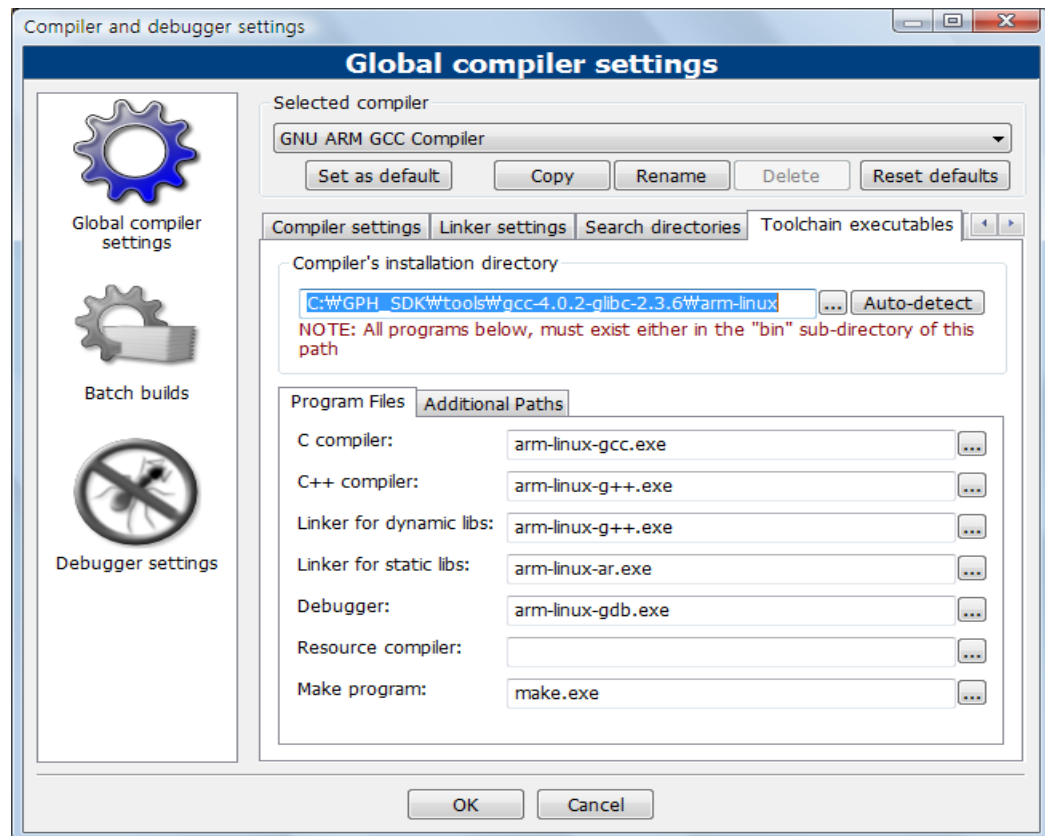
The above screenshot shows settings of host compiler. Host Compiler defaults to GCC Compiler that gets automatically installed when installing CodeBlocks. It's recommended to use the original setting as shown above. When CodeBlocks is installed in other directory, you can change "Compiler's installation directory" with the directory.

In order to change settings:

5. From "Settings" menu, select "Compiler and Debugger".
6. On "Selected compiler" list, select "GNU GCC Compiler".
7. From tab menu, select "Toolchain executables".
8. From sub-tab menu, select "Program Files".

3.2. Cross Compiler Settings

3.2.1. Compiler installation directory

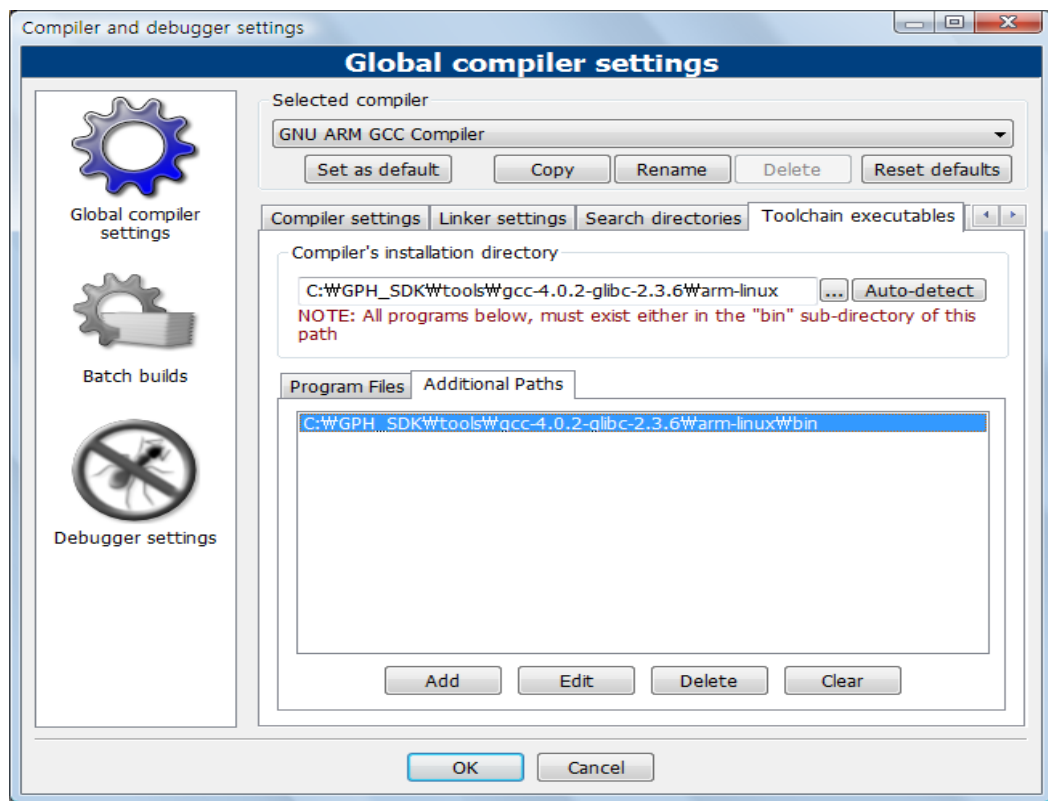


The above screenshot is for settings of Cross Compiler. As the Cross Compiler, GCC Cross Compiler included in GPH SDK shall be used. Assign the executable files and the directory where Cross Compiler is installed.

In order to change settings:

1. From "Settings" menu, select "Compiler and Debugger".
2. On "Selected compiler" list, select "GNU ARM GCC Compiler".
3. From tab menu, click on "Toolchain executables".
4. Change "Compiler's installation directory" to the directory where compiler is installed.
5. From sub-tab menu, select "Program Files".
6. Click on explore button to select the relevant executable file.

3.2.2. Additional Paths

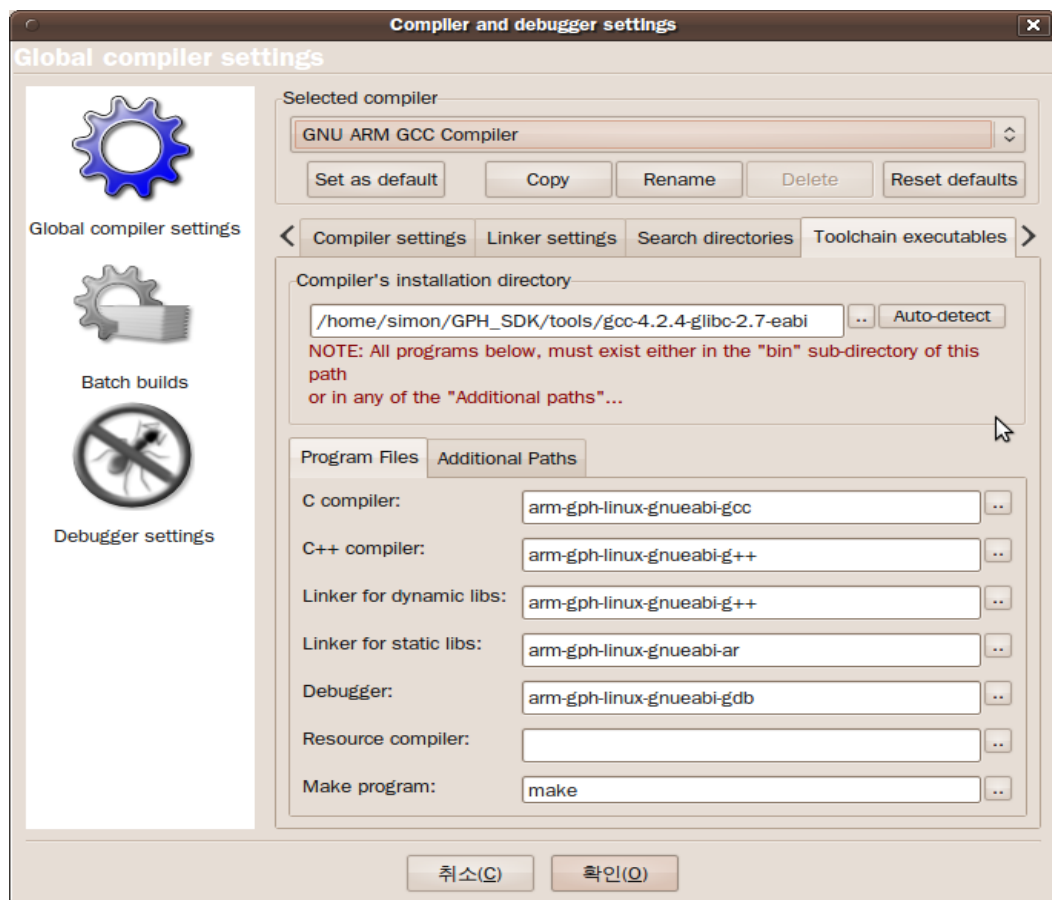


"Additional Paths" assigns an extra path that is needed when compiling or executing a program. The above case shows how the paths are assigned for the directories containing Window DLL files required to compile or execute a program.

In order to change settings:

1. From "Settings" menu, select "Compiler and Debugger".
2. On "Selected compiler" list, select "GNU ARM GCC Compiler".
3. From tab menu, select "Toolchain executables".
4. From sub-tab menu, select "Additional Paths".
5. Click on "Add" button to add a directory.

3.2.3. Compiler installation directory (Linux)



The above screenshot is for settings of Cross Compiler. As the Cross Compiler, GCC Cross Compiler included in GPH SDK shall be used. Assign the executable files and the directory where Cross Compiler is installed.

In order to change settings:

7. From "Settings" menu, select "Compiler and Debugger".
8. On "Selected compiler" list, select "GNU ARM GCC Compiler".
9. From tab menu, click on "Toolchain executables".
10. Change "Compiler"s installation directory" to the directory where compiler is installed.
11. From sub-tab menu, select "Program Files".
12. Click on explore button to select the relevant executable file.

4. New Project

4.1. Creating Project

1) For Window OS

Templates for create new project will be automatically installed at your installed codeblocks directory.

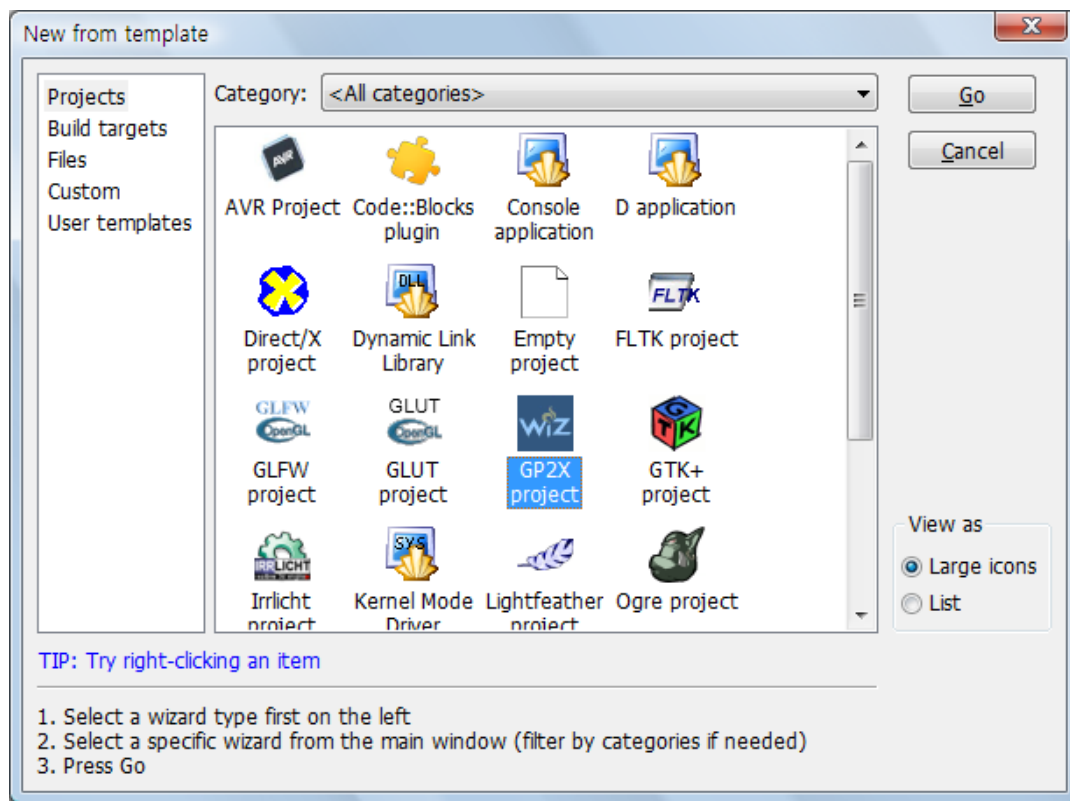
2) For Linux OS

Copy template files to the codeblocks directory manually.

This will update config.script, and copy gp2x templates to the codeblocks.

```
# cp GPH_SDK/tools/codeblocks/wizard/* /usr/share/codeblocks/templates/wizard
```

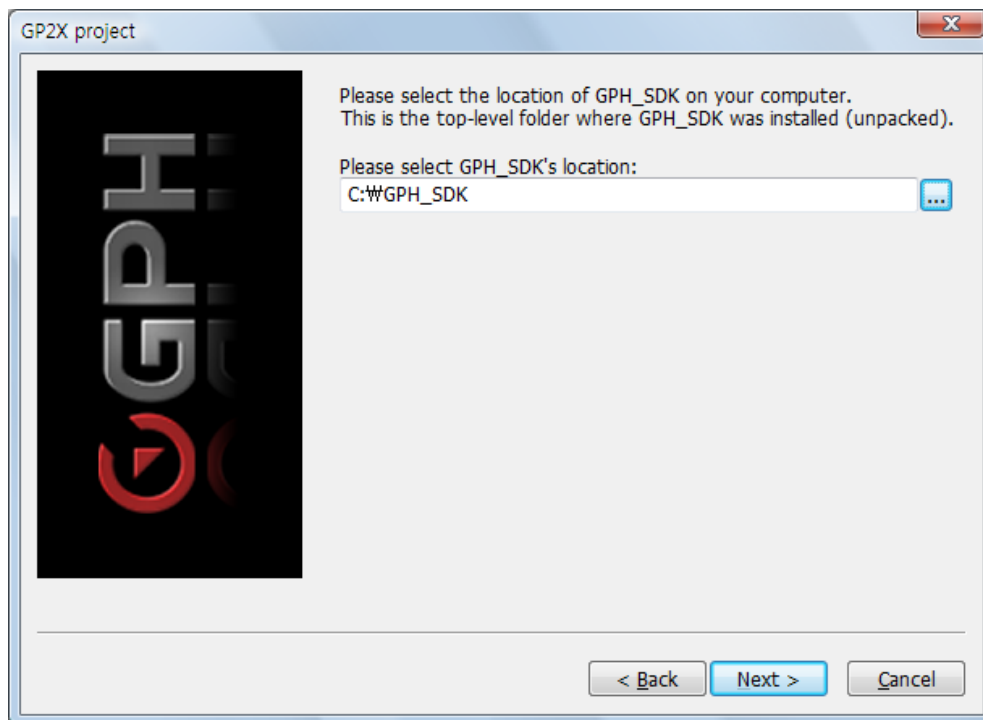
4.2. Using Templates



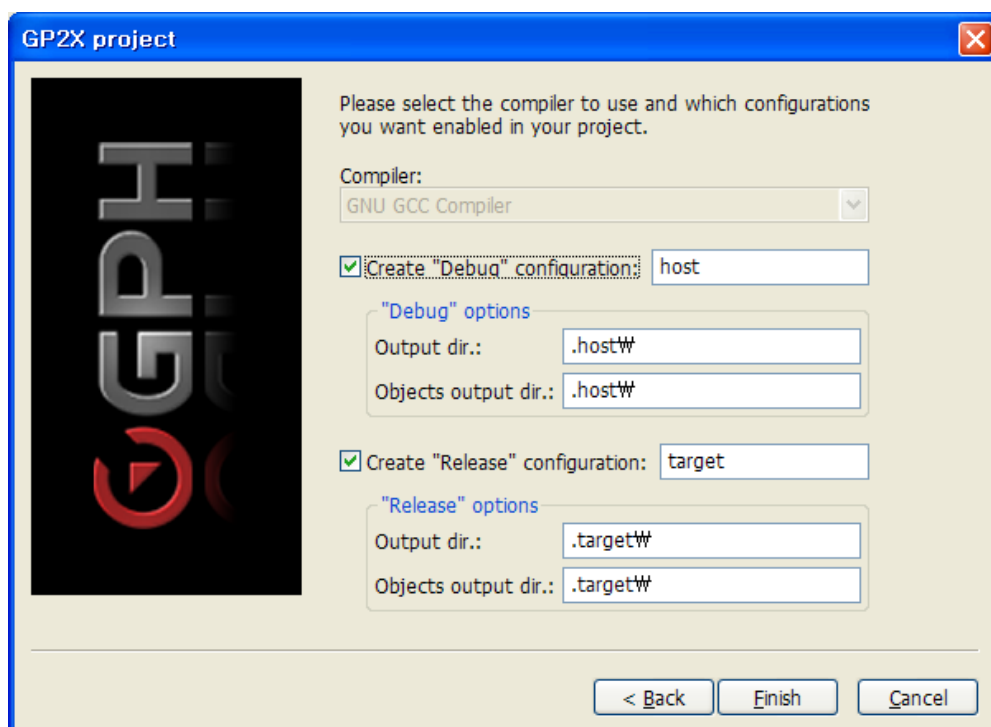
GPH SDK includes templates that allow you to create a project easily. There are two sets of templates available in GPH SDK. One is for game software development and the other is for application development. These templates get automatically installed when installing GPH SDK. If you want to manually install templates that you need, please refer to GPH_SDK/tools/codeblocks directory. ₩

In order to create a project using template:

1. From "File" menu, select "New" -> "Project" .
2. From the list in left, select "Projects".
3. Click "GP2X project" icon.
4. Follow the guides of project creation wizard to create a project.



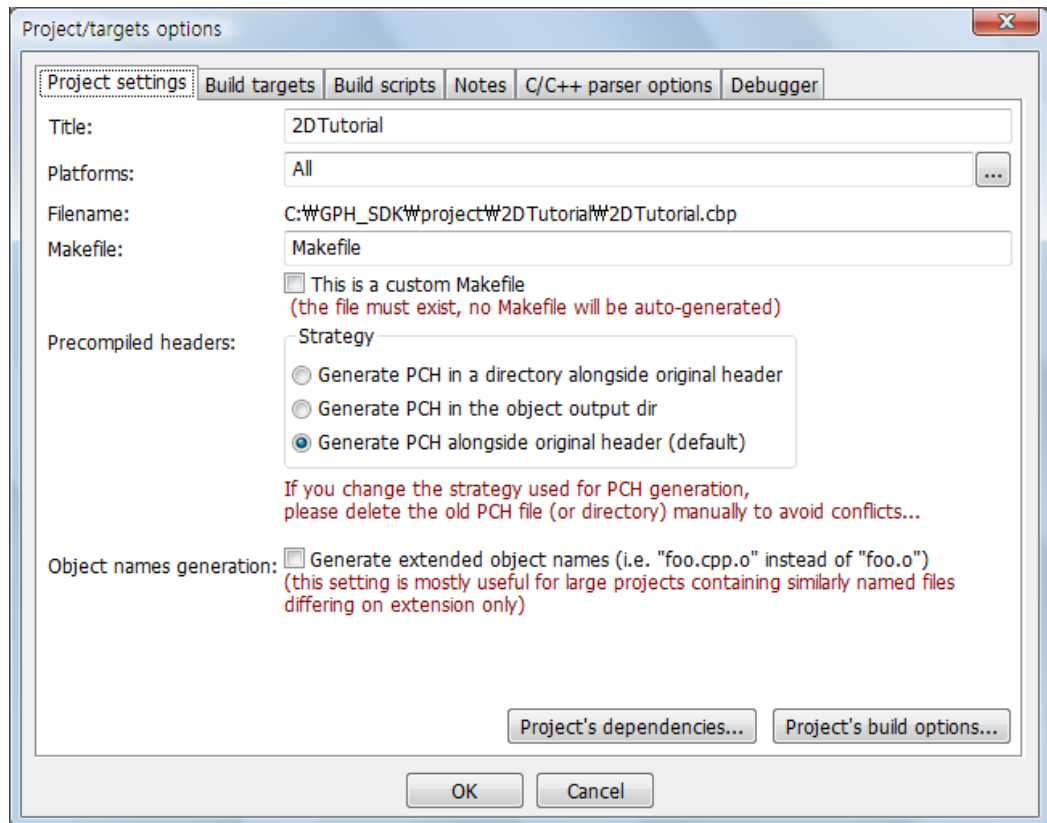
Assign the directory where GPH SDK is installed as shown above.



Assign host and target compile option as shown above.

5. Project/target options

5.1. Project settings



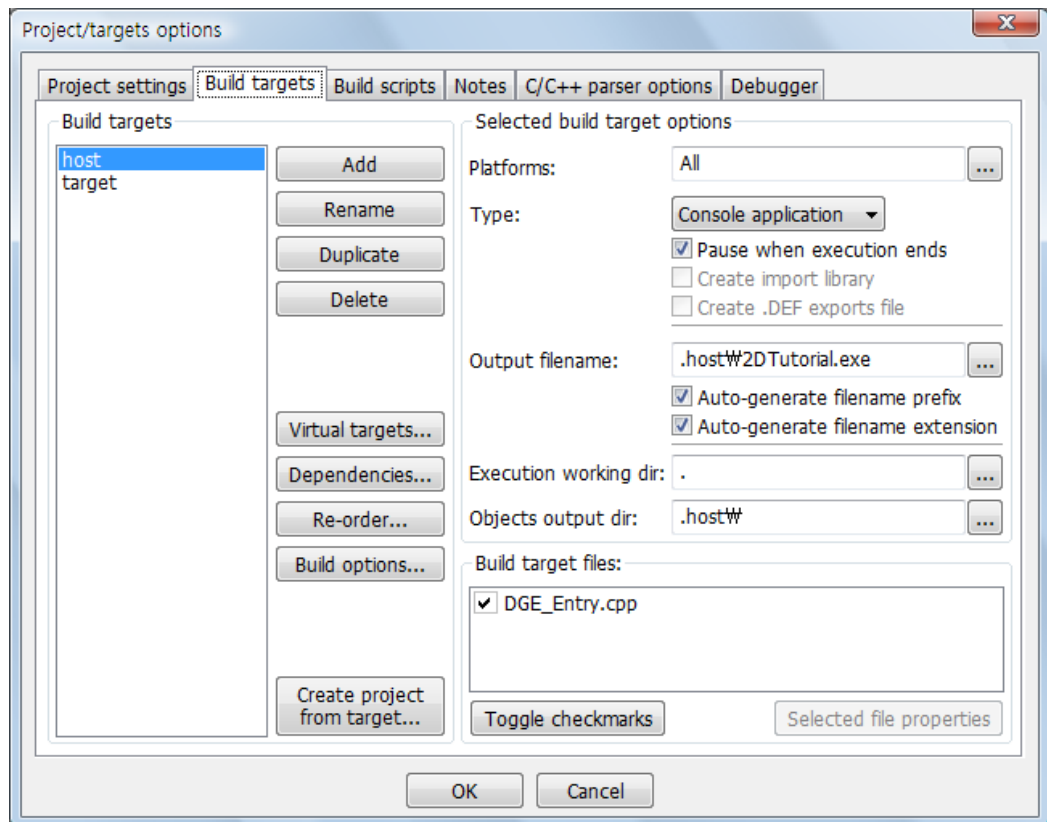
Above screenshot shows how to set properties of project. If you are using external Makefile, select "This is custom Makefile".

In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Properties".
3. From tab menu, click "Project settings".

5.2. Build targets options

5.2.1. Build targets -> host

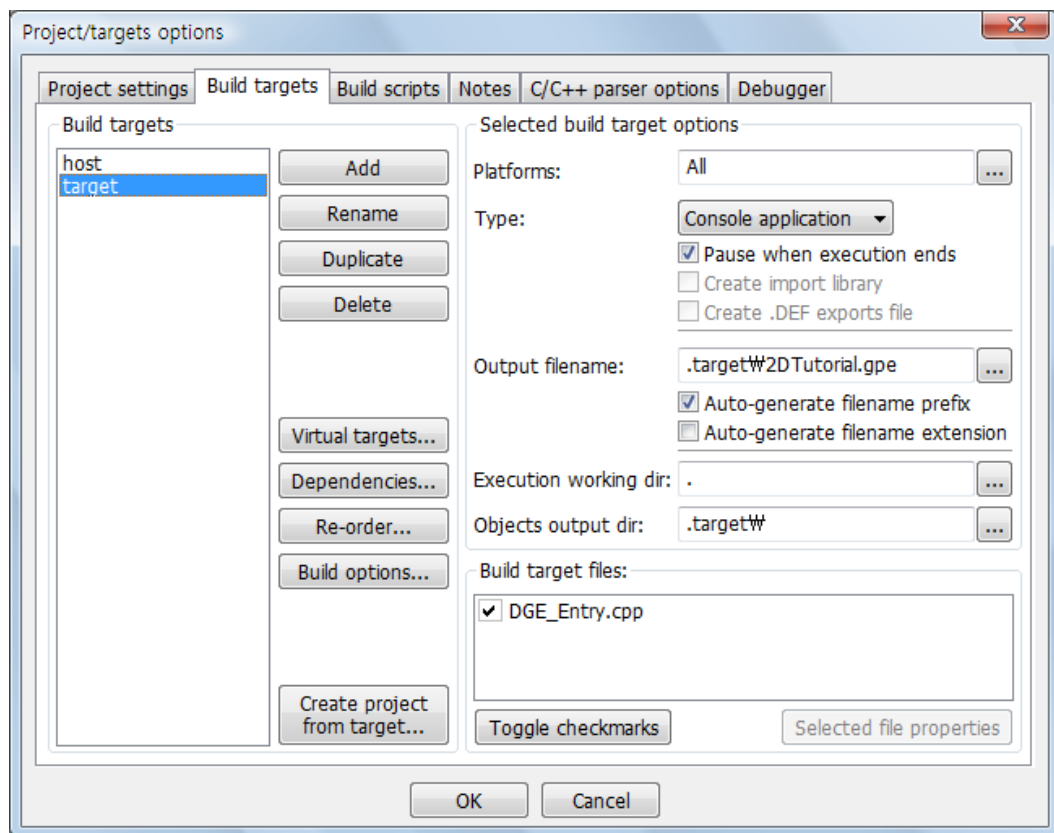


Above screenshot shows how to set a build target when host-compiling. Object files and executable files get created under ".host" directory and it can be changed with other directory when needed. All executable files are set to be created with ".exe" extension.

In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Properties".
3. From tab menu, select "Build targets".
4. On "Build targets" list, select "host".

5.2.2. Build targets -> target



Above screenshot shows how to set a build target when cross-compiling. Object files and executable files get created under ".target" directory and it can be changed with other directory when needed. All executable files are set to be created with ".gpe" extension.

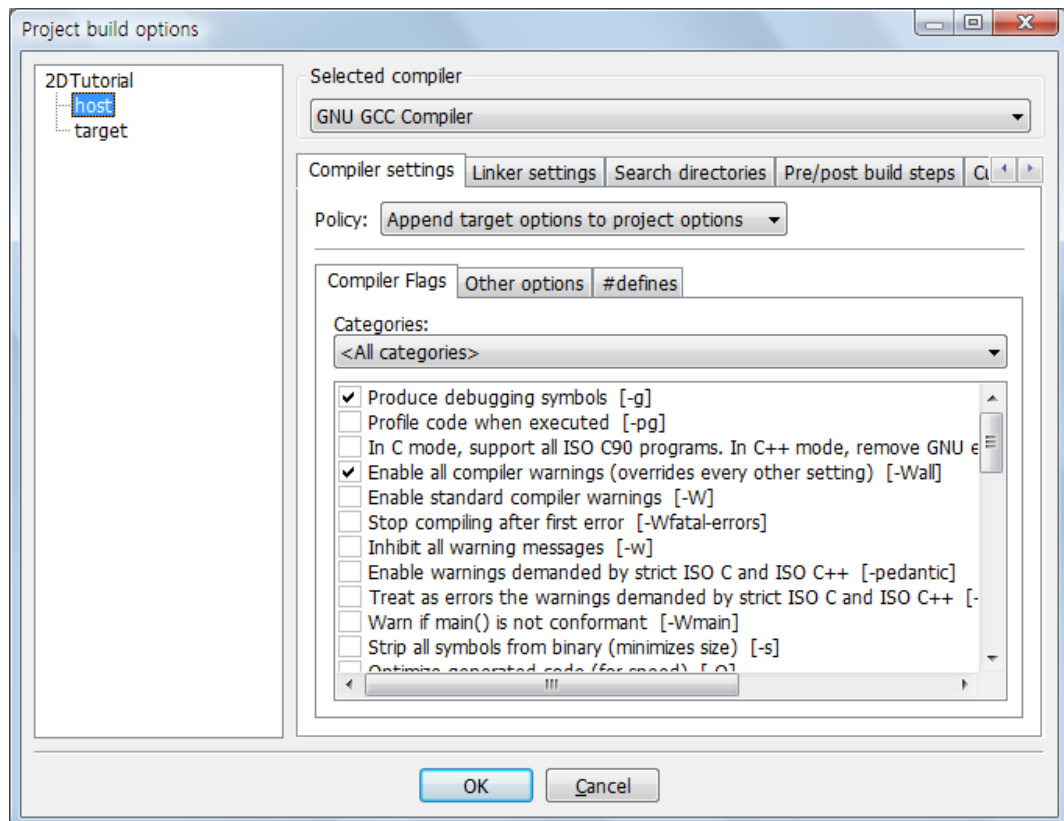
In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Properties".
3. From tab menu, select "Build targets".
4. On "Build targets" list, select "target".

5.3. Project build options

5.3.1. Project build options (host)

5.3.1.1. Project build options -> Compiler settings -> Compiler flags

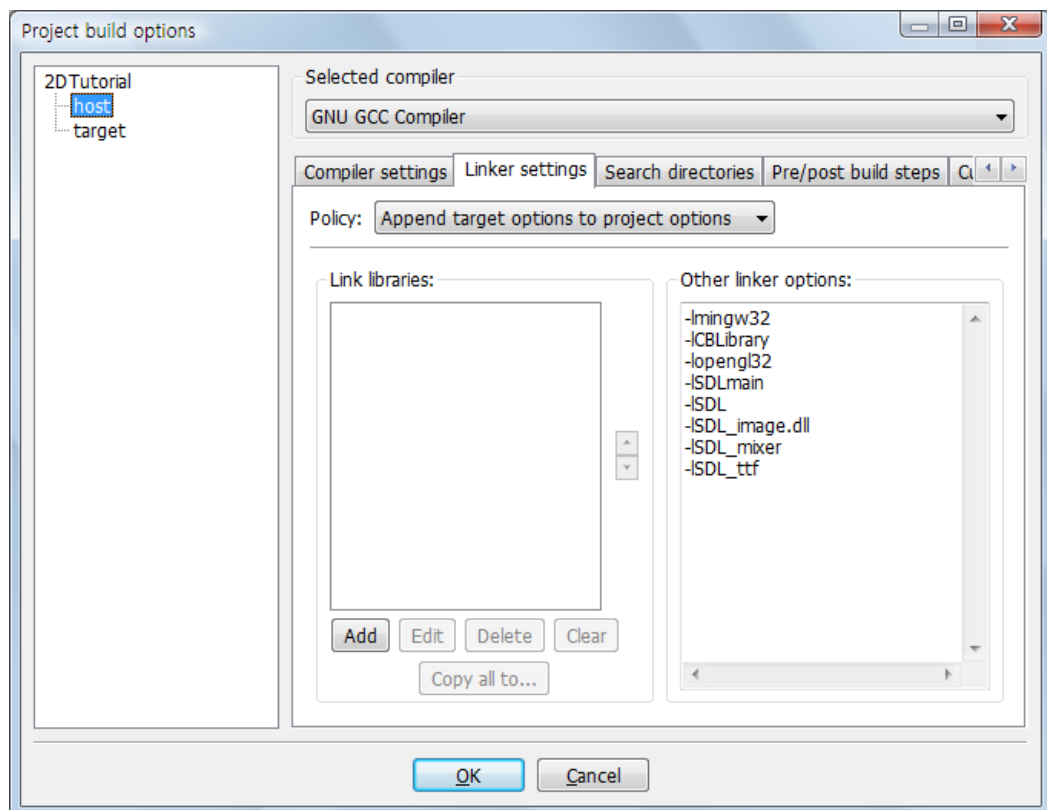


Above screenshot shows how to set "Compiler Flags" from the compile option of host compiler.

In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Build options".
3. From the list in left, select "host".
4. On "Selected compiler" list, select "GNU GCC Compiler".
5. From tab menu, click "Compiler settings".
6. From sub-tab menu, click "Compiler flags".

5.3.1.2. Project build options -> Linker settings -> Other linker options

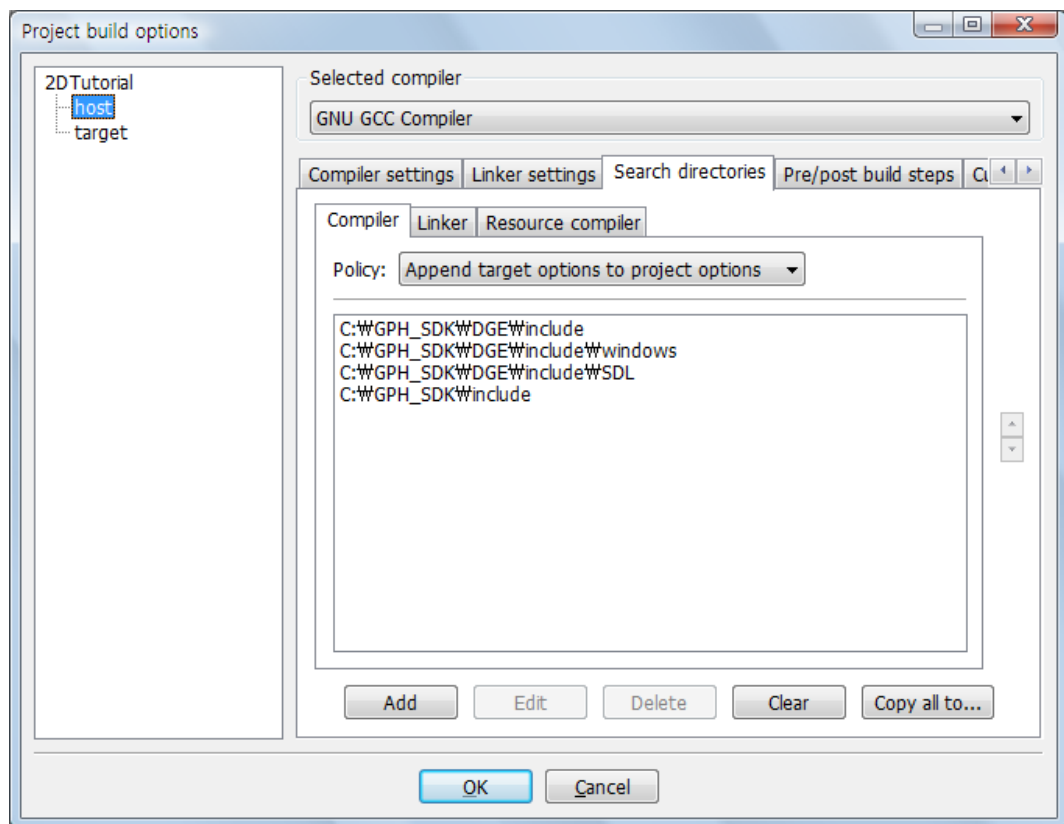


This screenshot shows how to set libraries from linker option of host compiler.

In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Build options".
3. From the list in left, select "host".
4. On "Selected compiler" list, select "GNU GCC Compiler".
5. From tab menu, click "Linker settings".
6. Click "Add" button to add a library or you may directly input from linker options.

5.3.1.3. Project build options -> Search directories -> Compiler

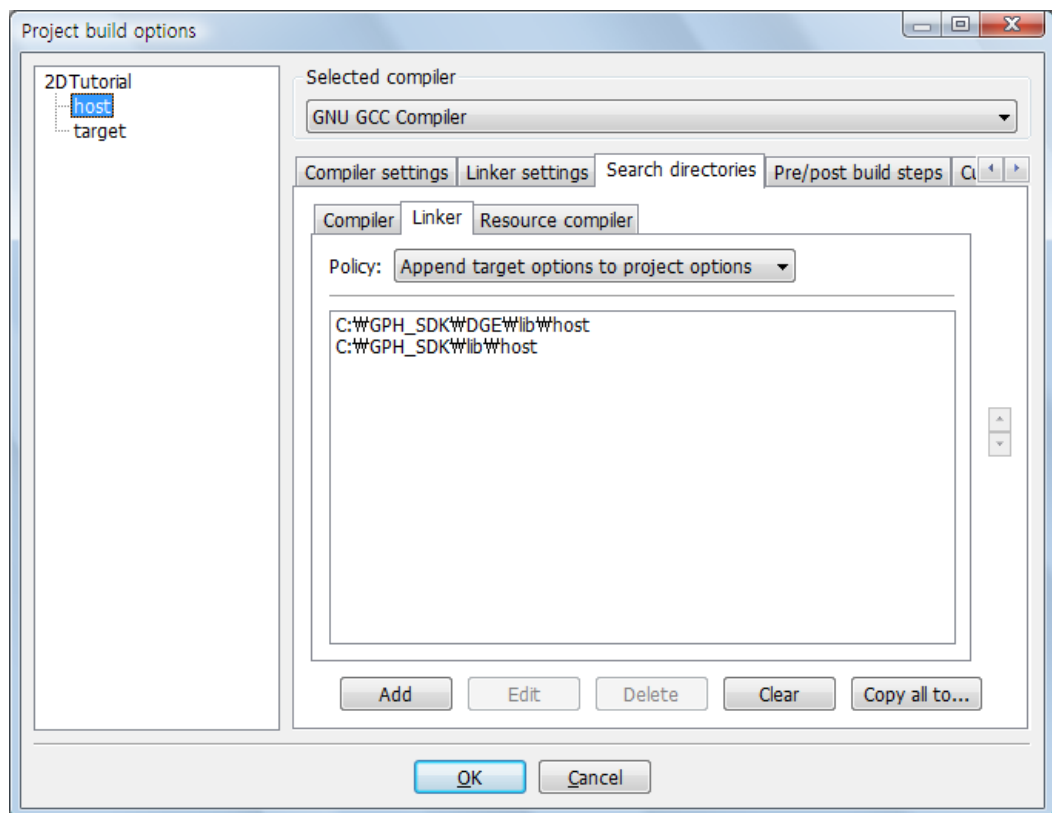


This screenshot shows how to set a path to a header file from compile option of host compiler.

In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Build options".
3. From the list in left, select "host".
4. On "Selected compiler" list, select "GNU GCC Compiler".
5. From tab menu, click "Search directories".
6. From sub-tab menu, click "Compiler".
7. Click "Add" button to add a path to a header file.

5.3.1.4. Project build options -> Search directories -> Linker



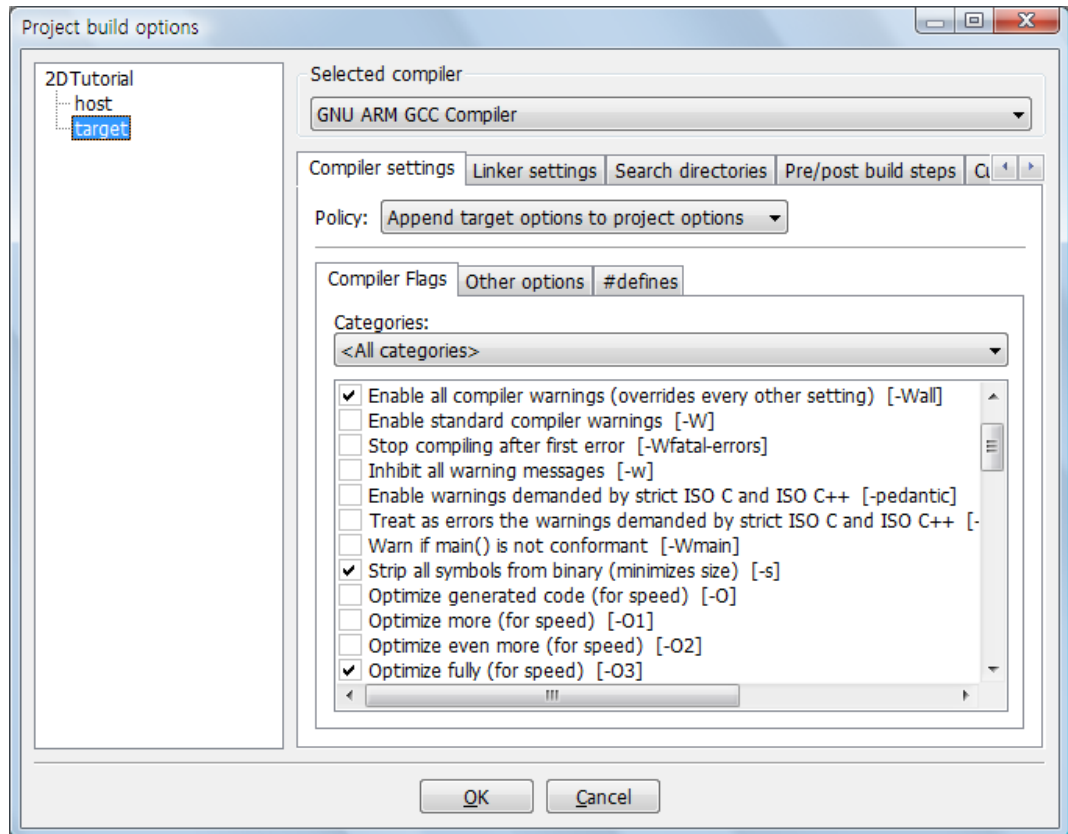
This screenshot shows how to set the path of libraries from linker option of host compiler.

In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Build options".
3. From the list in left, select "host".
4. From "Selected compiler" list, select "GNU GCC Compiler".
5. From tab menu, click "Search directories".
6. From sub-tab menu, click "Linker".
7. Click "Add" button to add a path to library.

5.3.2. Project build options (target)

5.3.2.1. Project build options -> Compiler settings -> Compiler Flags

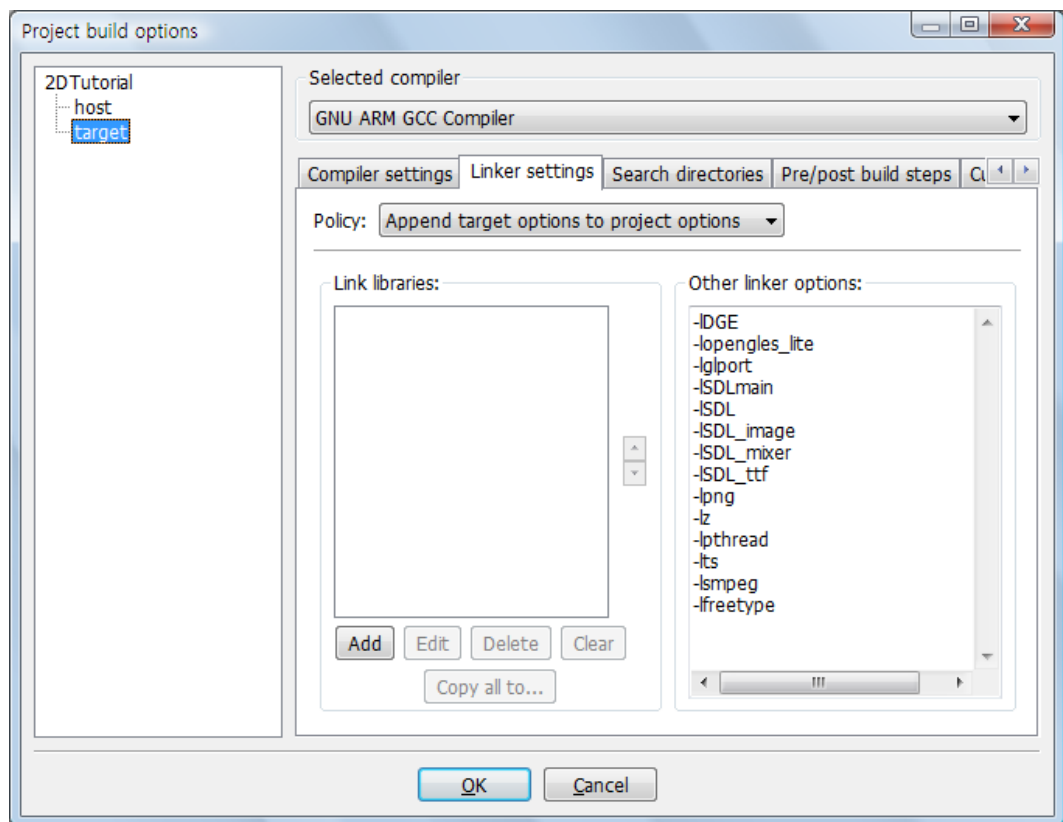


The above screenshot shows how to set "Compiler Flags" from the compile option of cross compiler.

In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Build options".
3. From the list in left, select "target".
4. On "Selected compiler" list, select "GNU ARM GCC Compiler".
5. From tab menu, click "Compiler settings".
6. From sub-tab menu, click "Compiler flags".

5.3.2.2. Project build options -> Linker settings -> Other linker options

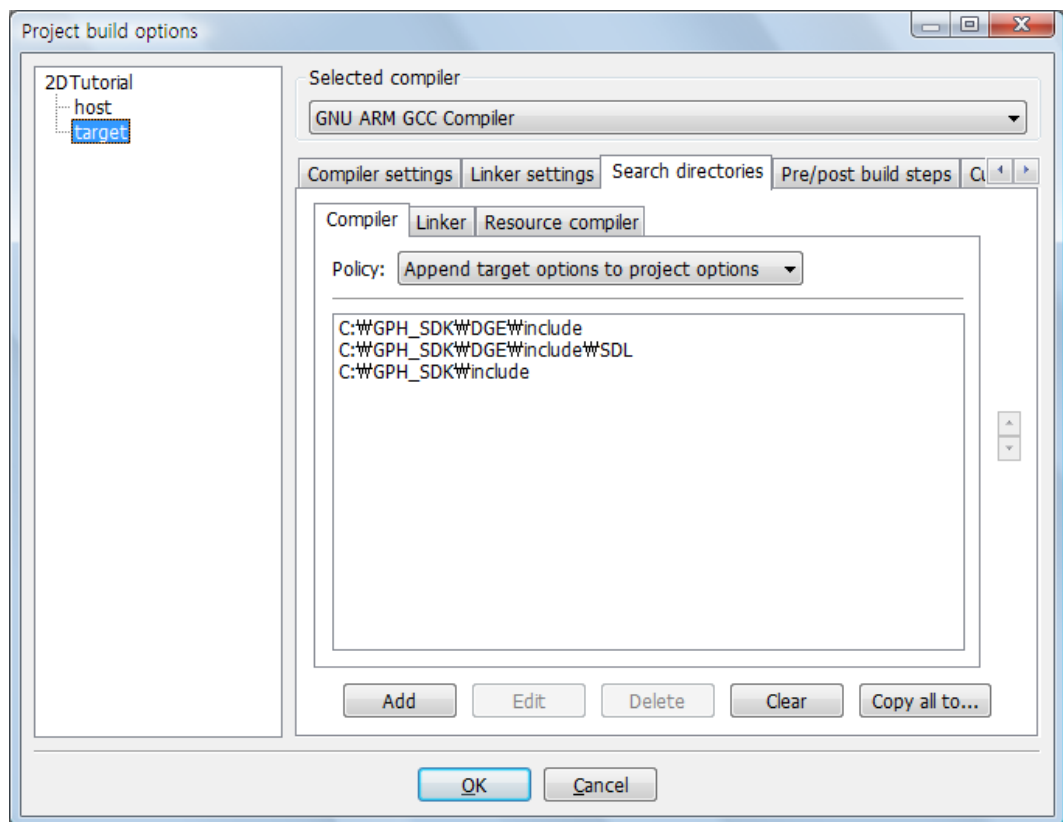


This screenshot shows how to set libraries from linker option of cross compiler.

In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Build options".
3. From the list in left, select "target".
4. On "Selected compiler" list, select "GNU ARM GCC Compiler".
5. From tab menu, click "Linker settings".
6. Click "Add" button to add a library or you may directly input from linker options.

5.3.2.3. Project build options -> Search directories -> Compiler

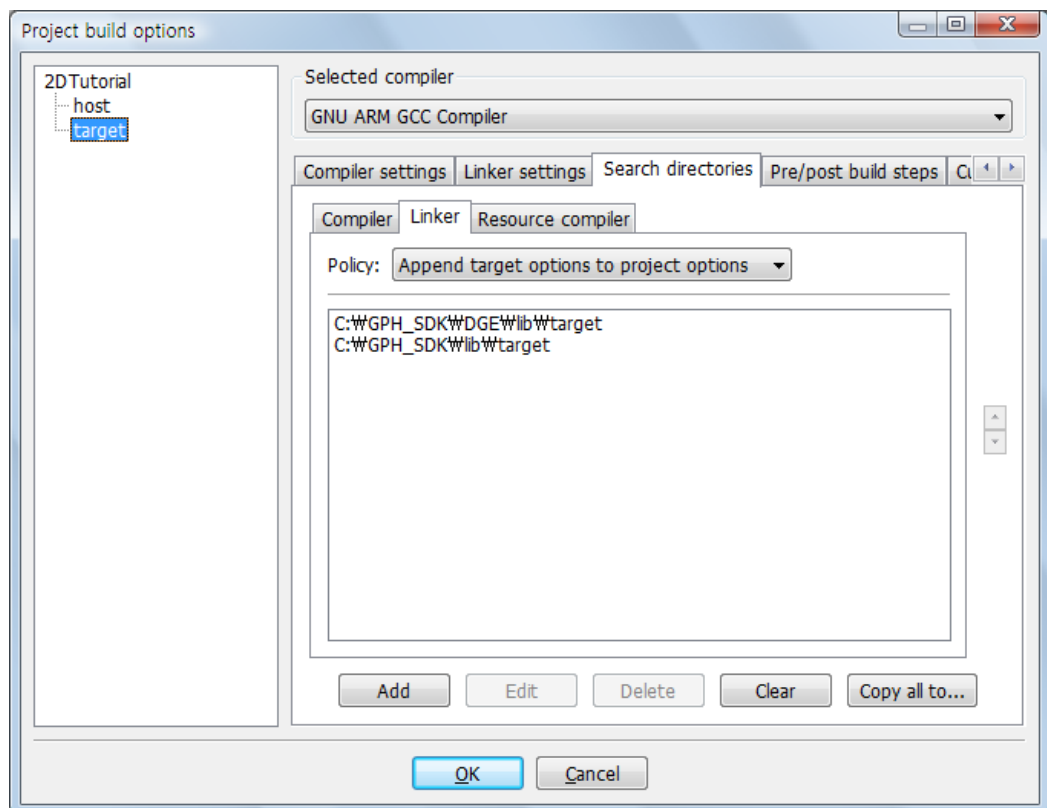


This screenshot shows how to set a path to a header file from compile option of cross compiler.

In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Build options".
3. From the list in left, select "target".
4. On "Selected compiler" list, select "GNU ARM GCC Compiler".
5. From tab menu, click "Search directories".
6. From sub-tab menu, click "Compiler".
7. Click "Add" button to add a path to a header file.

5.3.2.4. Project build options -> Search directories -> Linker

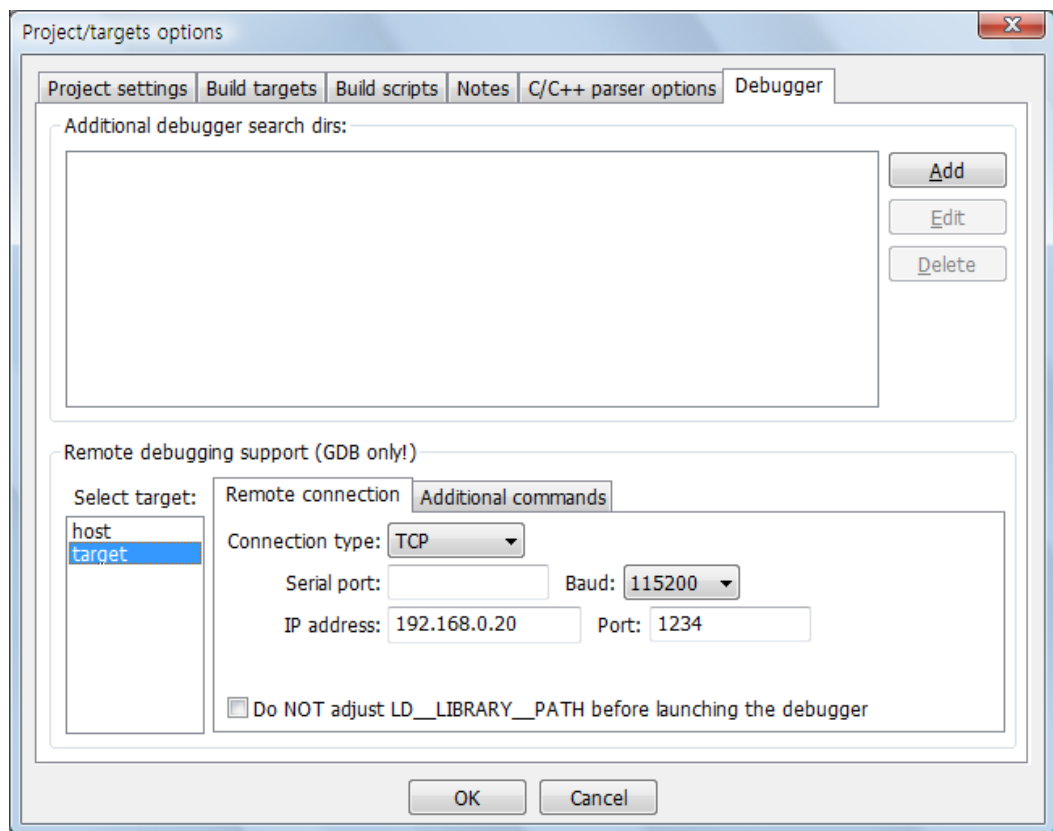


This screenshot shows how to set the path of libraries from linker option of cross compiler.

In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Build options".
3. From the list in left, select "target".
4. From "Selected compiler" list, select "GNU ARM GCC Compiler".
5. From tab menu, click "Search directories".
6. From sub-tab menu, click "Linker".
7. Click "Add" button to add a path to library.

5.4. Debugger



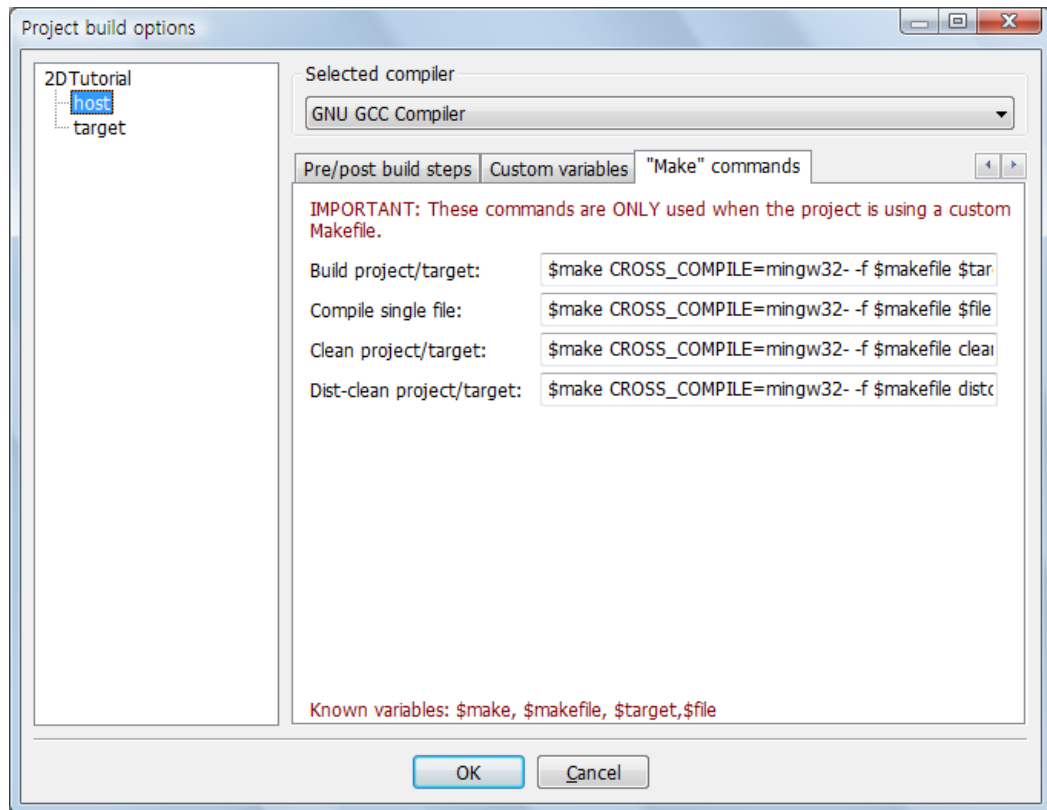
The above screenshot shows the setting for remote debugging. The remote debugging only works when host machine and target machine are connected through a network. Wi-Fi kernel module and tool for network connection can be found under GPH_SDK/tools directory. From those tools, you can assign IP address and port number of the target machine where gdbserver is running.

In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Properties".
3. From tab menu, click "Debugger".
4. On "Select target" list, select a target.

6. Using Makefile

6.1. Host "Make" commands

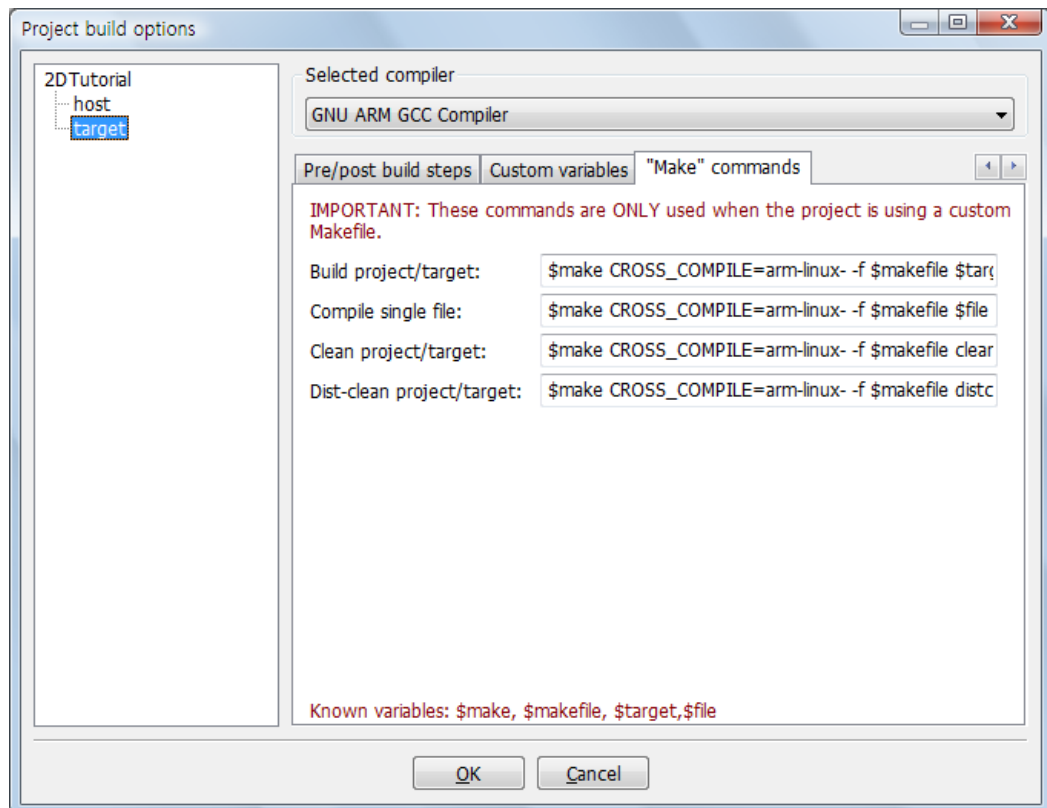


This screenshot shows the setting for host compile when using an external Makefile. To compile by using Makefile, please refer to the section 5.1 Project settings. It shows you how to set environmental variations of CROSS_COMPILE and use them on Makefile.

In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Build options".
3. From the list in left, select "host".
4. On "Selected compiler" list, select "GNU GCC Compiler".
5. From tab menu, click "Make commands".

6.2. Target "Make" commands



This screenshot shows how to set the options for cross compiling when using an external Makefile. To compile by using Makefile, please refer to the section 5.1 Project settings. It shows you how to set environmental variations of CROSS_COMPILE and use them on Makefile.

In order to change settings:

1. Open a project or create a new one.
2. From "Project" menu, select "Build options".
3. From the list in left, select "target".
4. On "Selected compiler" list, select "GNU ARM GCC Compiler".
5. From tab menu, click "Make commands".

6.3. Makefile : example

Makefile is composed to be used on both host compile and cross compile. You can simply use the editor to edit or add source codes, header files or libraries.

```
TARGET    = 2DTutorial
OBS       = DGE_Entry.o

# for cygwin
# CROSS_COMPILE = arm-linux-

CC        = $(CROSS_COMPILE)gcc
CPP       = $(CROSS_COMPILE)g++
SDK_HOME  = C:/GPH_SDK

ifeq ($(CROSS_COMPILE),arm-linux-)
BIN       = .target/$(TARGET).gpe
INCS      = -I$(SDK_HOME)/DGE/include -I$(SDK_HOME)/DGE/include/SDL -I$(SDK_HOME)/include
CFLAGS    = -Wall -O3 -fexceptions -fomit-frame-pointer -fPIC
LIBS      = -L$(SDK_HOME)/DGE/lib/target -L$(SDK_HOME)/lib/target \W
           -ISDL -ISDL_image -ISDL_mixer -ISDL_ttf -IDGE -lglport -lopengles_lite -lpng -lz -lts -lsmpeg -lfreetype
LFLAGS    = -s
else
BIN       = .host/$(TARGET).exe
INCS      = -I$(SDK_HOME)/DGE/include -I$(SDK_HOME)/DGE/include/SDL -I$(SDK_HOME)/include
CFLAGS    = -g -Wall -fexceptions
LIBS      = -L$(SDK_HOME)/DGE/lib/host -L$(SDK_HOME)/lib/host \W
           -lmingw32 -IDGE -lopengl32 -ISDLmain -ISDL -ISDL_image -ISDL_mixer -ISDL_ttf
endif

.c.o:
    $(CC) $(INCS) $(CFLAGS) -c $< -o $@

.cpp.o:
    $(CPP) $(INCS) $(CFLAGS) -c $< -o $@

$(BIN): $(OBS)
    $(CPP) $(OBS) -o $@ $(LIBS) $(LFLAGS)

all host target: $(BIN)

clean cleanhost cleantarget:
```