

GPH SDK

Version 10.08

10.02	GPH_SDK for WIZ	
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

이 문서는 GPH SDK 를 설치하는 방법을 설명하고, GPH SDK 를 사용하는데 있어서 필요한 각종 설정에 대한 기본적인 내용을 설명하고 있습니다.

1.1. What is GPH SDK ?

GPH SDK 는 (주)게임파크홀딩스에서 개발한 휴대용 기기를 위한 소프트웨어 개발 툴킷입니다. 개발자들은 GPH SDK 를 이용하여 다양한 게임 및 어플리케이션을 개발할 수 있습니다. GPH SDK 는 다음과 같이 크게 두 가지 요소로 구성되어 있습니다.

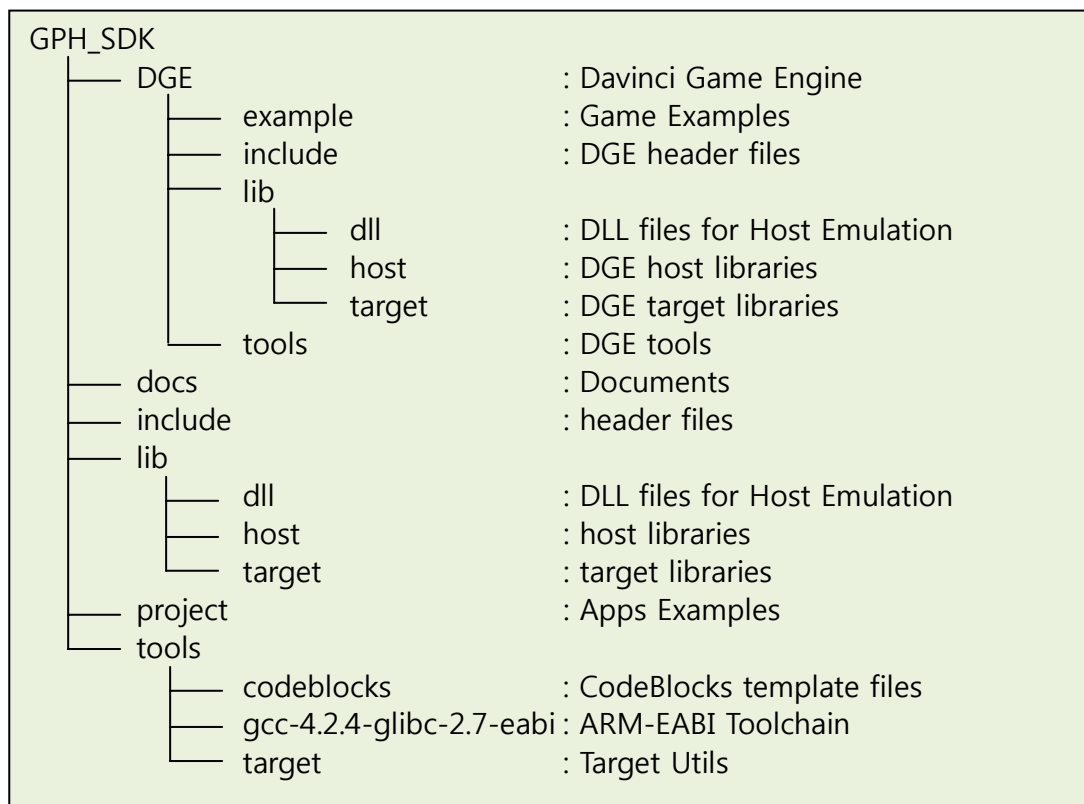
- 게임엔진 DGE (Davinci Game Engine)
- 각종 라이브러리들과 헤더파일

GPH SDK 는 다양한 통합개발환경을 지원합니다. 이 문서는 Code::Blocks 를 사용하는 경우를 예를 들어 설명하였습니다. Visual Studio 또는 Visual C++ 사용자는 별도의 문서를 참고하시기 바랍니다. 또한, GPH SDK 는 두 가지 종류의 배포판을 제공합니다.

- 윈도우 사용자용 배포판
- 리눅스 사용자용 배포판

GPH SDK 는 호스트 컴파일 및 크로스 컴파일을 동시에 지원하여, 편리하게 게임 및 어플리케이션을 개발할 수 있는 환경을 제공합니다. 또한 호스트 에뮬레이션 및 크로스 디버깅을 동시에 지원하여 여러 가지 환경에서 소프트웨어 디버깅을 할 수 있는 방법을 지원합니다.

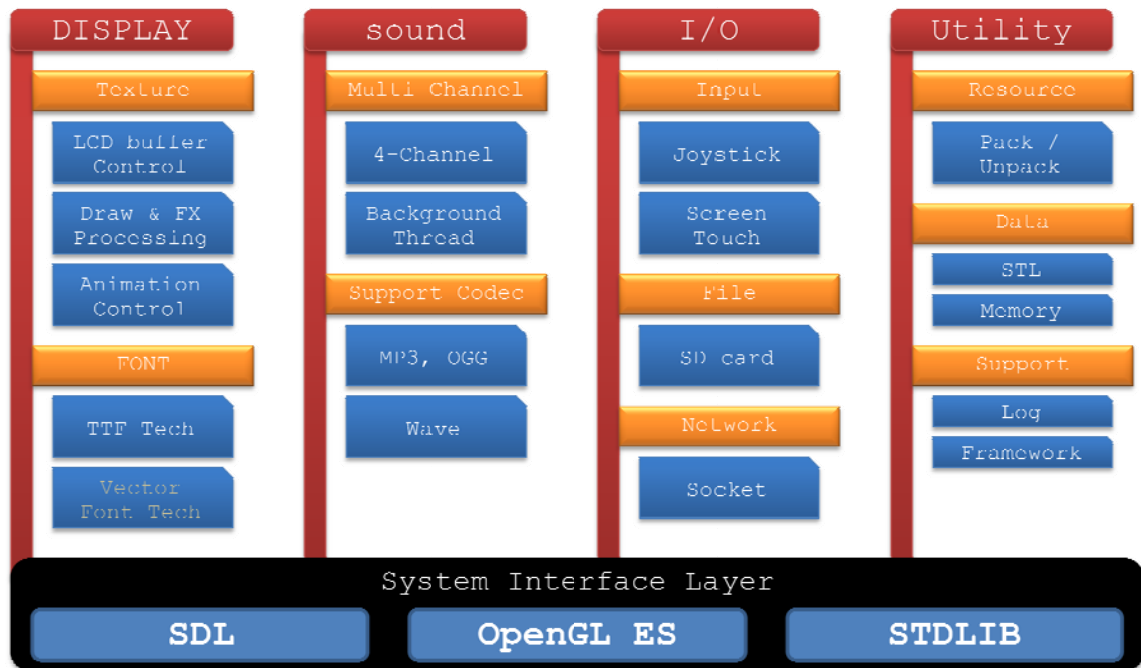
GPH SDK의 디렉토리 구성은 다음과 같습니다.



<Fig 1.1> Directory Structure

1.2. What is DGE ?

DGE 는 “Davinci Game Engine”의 약어이며 (주)게임파크홀딩스에서 개발한 게임기기들을 위한 게임엔진 입니다. DGE 는 크게 SDL 과 OpenGL 을 기반으로 하고 있으며, 다양한 게임들을 쉽고 편하게 개발할 수 있도록 해 줍니다. DGE 의 대략적인 구성은 다음과 같습니다.



<Fig 1.2> Davinci Architecture

2. Installing SDK

2.1. Installing CodeBlocks

CodeBlocks 은 오픈소스, 멀티플랫폼을 지원하는 통합개발환경입니다. 아래의 인터넷 주소에서 윈도우 또는 리눅스등 자신이 사용하는 플랫폼에 해당되는 최신의 설치파일을 다운받아 설치하시기 바랍니다.

<http://www.codeblocks.org>

1) 윈도우 OS 인 경우

윈도우 OS 를 사용하는 경우, 크로스 컴파일러로써 CodeBlocks 에 내장되어 있는 MinGW 를 사용하므로, MinGW 를 포함하고 있는 CodeBlocks 설치파일을 다운받아서 설치하시기 바랍니다.

2) 리눅스 OS 인 경우

```
[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

3) 윈도우 OS 인 경우

- 설치파일인 GPH_SDK.exe 를 실행하여 GPH SDK 를 설치합니다.
- 이 문서는 C:\GPH_SDK 의 디렉토리에 설치된 경우로 설명되어 있습니다.

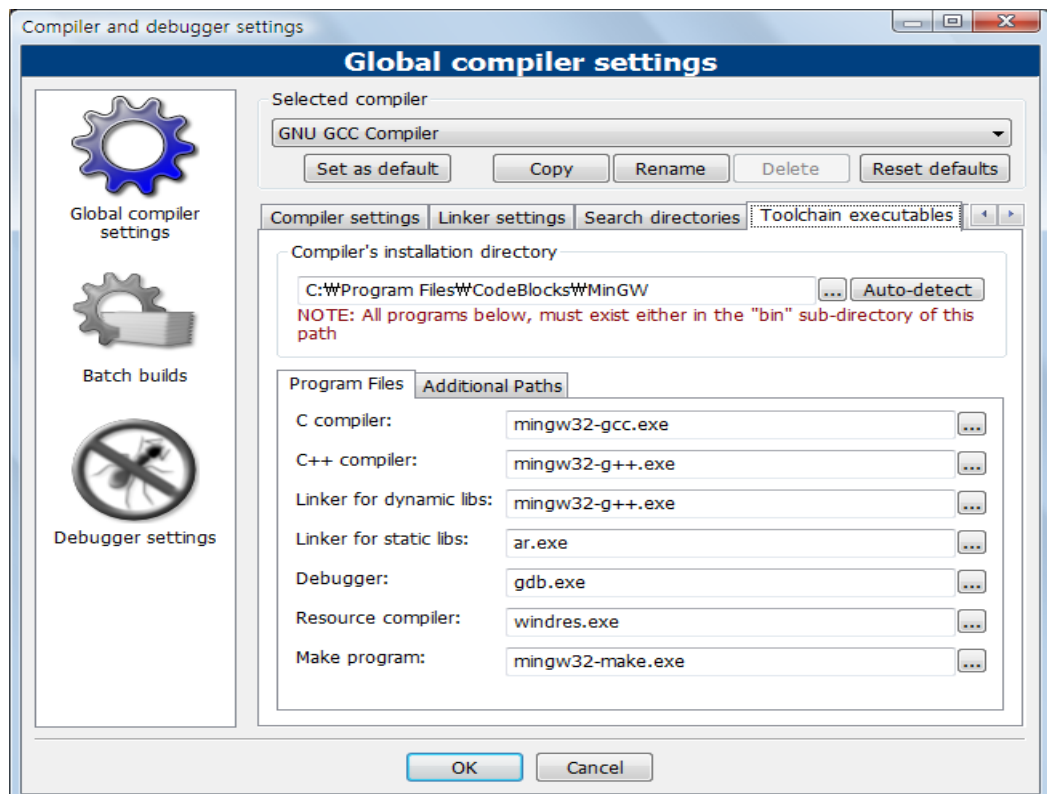
4) 리눅스 OS 인 경우

- 설치파일인 GPH_SDK.tar.gz 파일을 홈디렉토리에 압축해제 합니다.

3. Compiler Settings

3.1. Native Compiler Settings

3.1.1. Compiler's installation directory

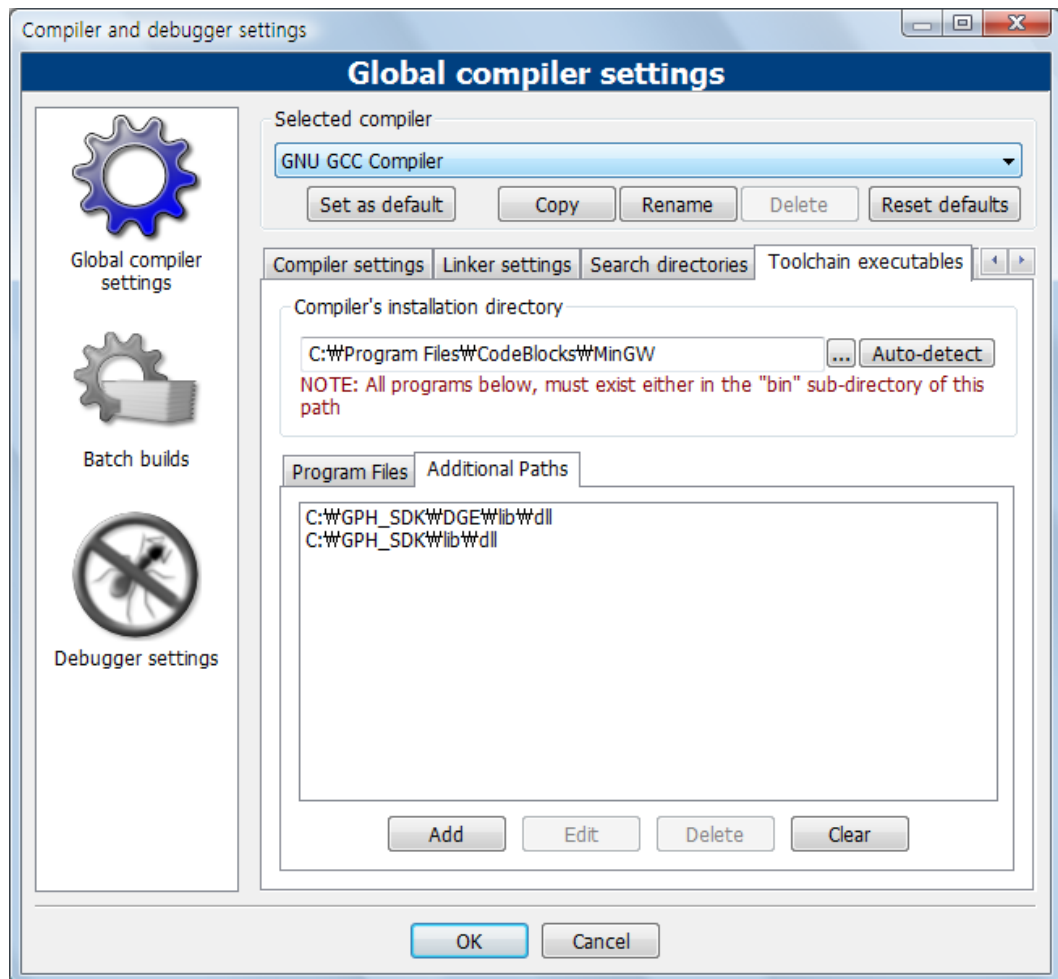


위 그림은 호스트 컴파일러에 관한 설정입니다. 호스트 컴파일러는 CodeBlocks를 설치할 때 기본적으로 설치되는 MinGW 컴파일러를 사용하게 됩니다. 설치할 때의 기본설정을 그대로 사용합니다. 만일 CodeBlocks가 다른 디렉토리에 설치되어 있는 경우에는 “Compiler’s installation directory”를 해당 디렉토리로 변경할 수 있습니다.

선택설정을 변경하려면,

1. “Settings” 메뉴에서 “Compiler and Debugger” 리스트를 선택합니다.
2. “Selected compiler”리스트에서 “GNU GCC Compiler”를 선택합니다.
3. 탭에서 “Toolchain executables”를 누릅니다.
4. 아래 탭에서 “Program Files”를 누릅니다.

3.1.2. Additional Paths

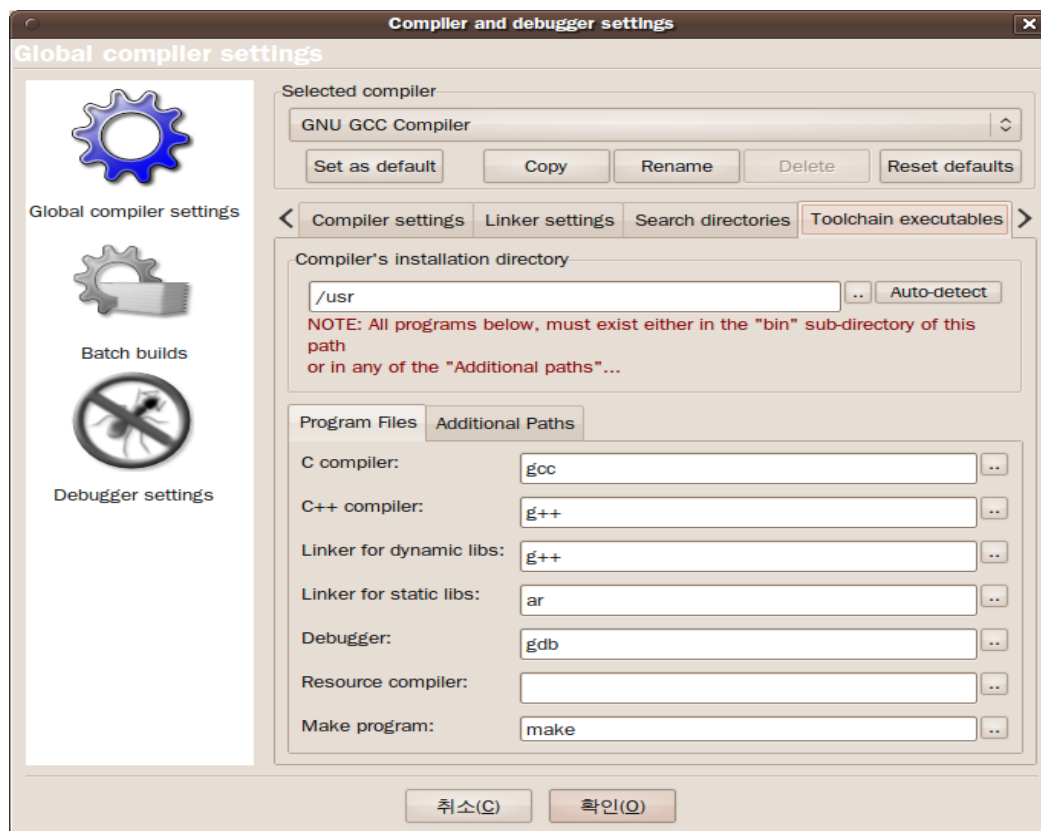


"Additional Paths" 는 프로그램이 컴파일 및 실행될 때 추가로 필요한 경로를 지정해 줍니다. 위의 경우는 호스트 컴파일 및 실행 시에 필요한 윈도우 DLL파일들이 있는 경로를 추가해준 것입니다.

선택설정을 변경하려면,

1. "Settings" 메뉴에서 "Compiler and Debugger" 리스트를 선택합니다.
2. "Selected compiler"리스트에서 "GNU GCC Compiler"를 선택합니다.
3. 탭에서 "Toolchain executables"를 누릅니다.
4. 아래 탭에서 "Additional Paths"를 누릅니다.
5. "Add" 버튼을 눌러 디렉토리를 추가해 줍니다.

3.1.3. Compiler installation directory (Linux)



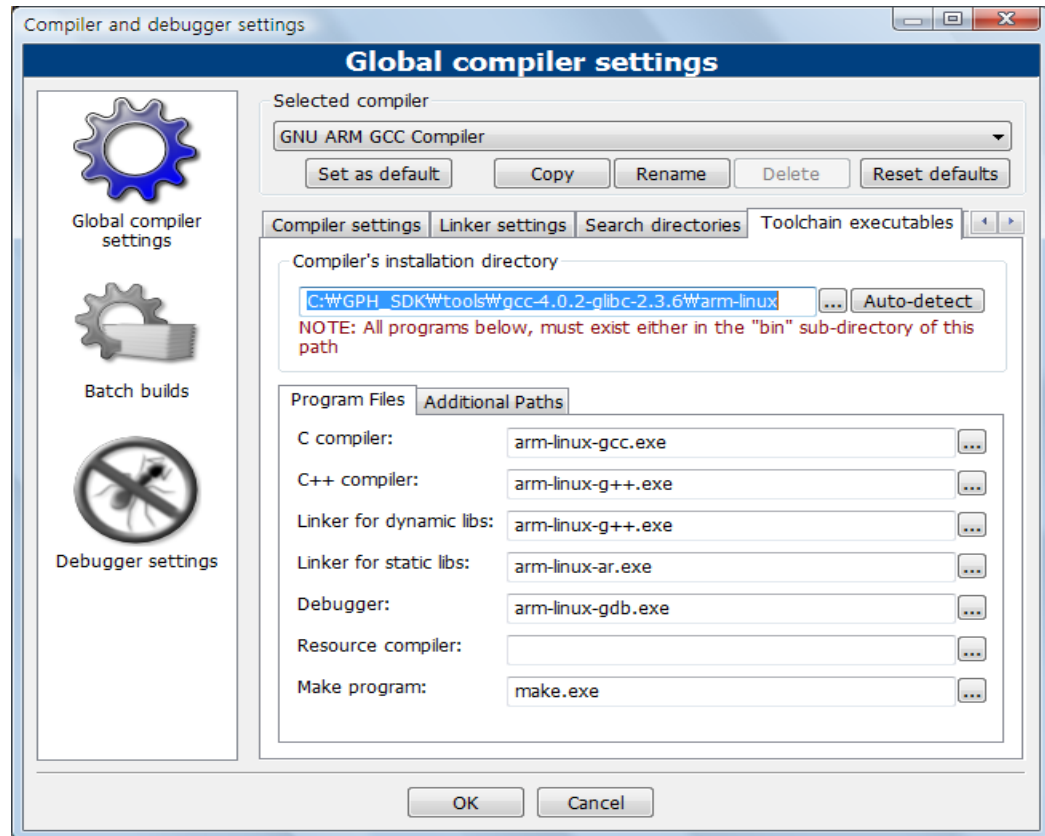
위 그림은 리눅스 사용자를 위한 호스트 컴파일러에 관한 설정입니다. 호스트 컴파일러는 CodeBlocks를 설치할 때 기본적으로 설치되는 GCC 컴파일러를 사용하게 됩니다. 설치할 때의 기본설정을 그대로 사용합니다. 만일 CodeBlocks가 다른 디렉토리에 설치되어 있는 경우에는 "Compiler's installation directory"를 해당 디렉토리로 변경할 수 있습니다.

선택설정을 변경하려면,

1. "Settings" 메뉴에서 "Compiler and Debugger" 리스트를 선택합니다.
2. "Selected compiler"리스트에서 "GNU GCC Compiler"를 선택합니다.
3. 탭에서 "Toolchain executables"를 누릅니다.
4. 아래 탭에서 "Program Files"를 누릅니다.

3.2. Cross Compiler Settings

3.2.1. Compiler's installation directory

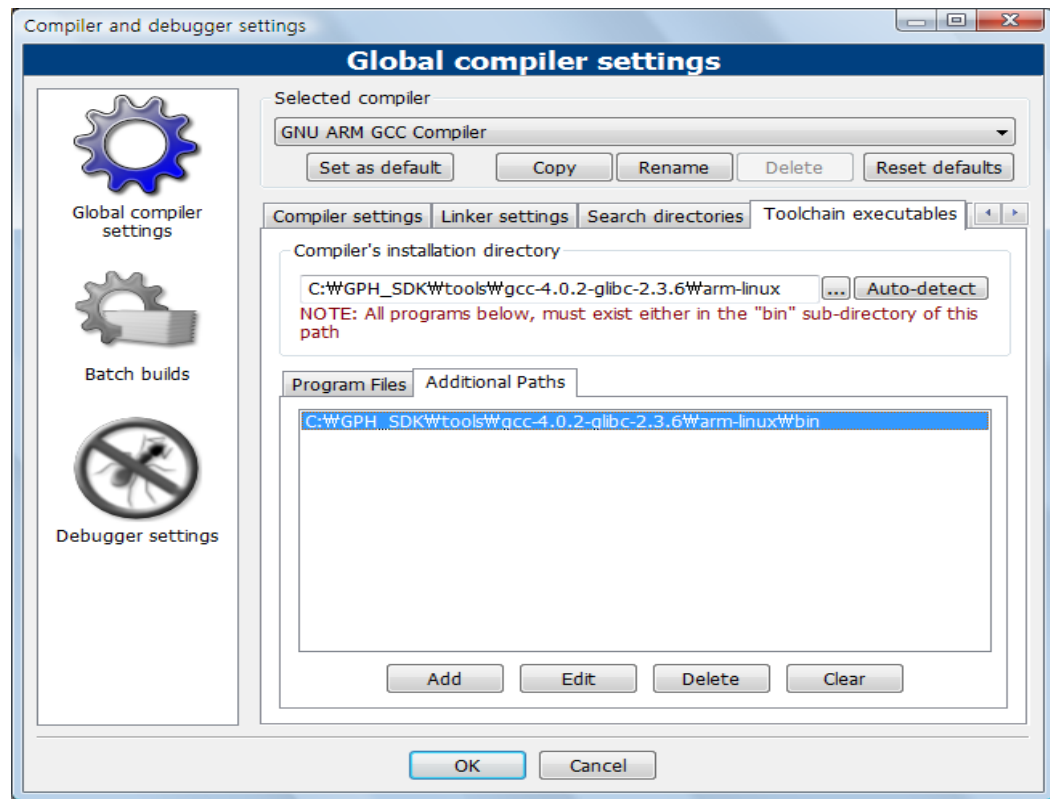


위 그림은 크로스 컴파일러에 관한 설정입니다. 크로스 컴파일러는 GPH SDK에 포함되어 있는 GCC Cross Compiler를 사용합니다. 크로스 컴파일러가 설치되어 있는 디렉토리를 지정해주고, 실행파일들을 지정해 줍니다.

선택설정을 변경하려면,

1. "Settings" 메뉴에서 "Compiler and Debugger" 리스트를 선택합니다.
2. "Selected compiler"리스트에서 "GNU ARM GCC Compiler"를 선택합니다.
3. 탭에서 "Toolchain executables"를 누릅니다.
4. "Compiler"s installation directory"를 컴파일러가 설치된 디렉토리로 변경합니다.
5. 아래 탭에서 "Program Files"를 누릅니다.
6. 탐색 버튼을 눌러서, 해당 실행 파일을 선택합니다.

3.2.2. Additional Paths

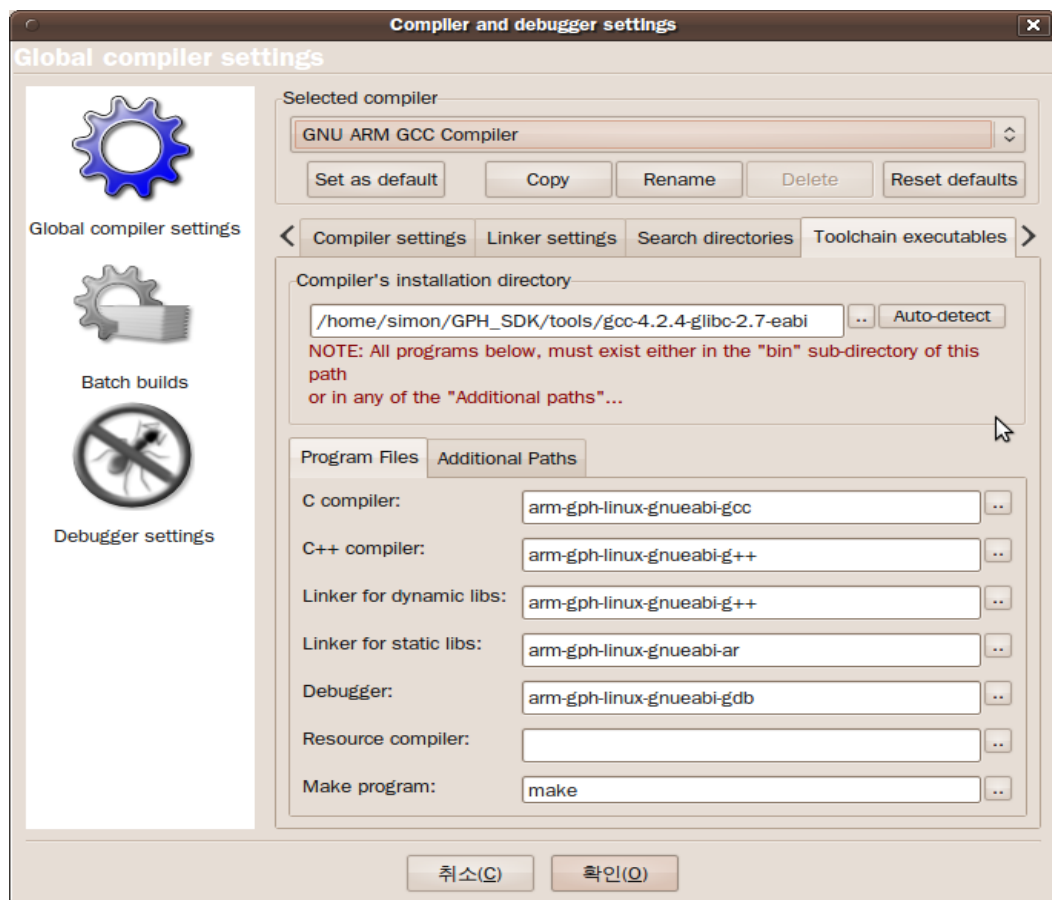


“Additional Paths” 는 프로그램이 컴파일 및 실행될 때 추가로 필요한 경로를 지정해 줍니다. 위의 경우는 호스트 컴파일 및 실행 시에 필요한 윈도우 DLL파일들이 있는 경로를 추가해준 것입니다.

선택설정을 변경하려면,

1. “Settings” 메뉴에서 “Compiler and Debugger” 리스트를 선택합니다.
2. “Selected compiler”리스트에서 “GNU ARM GCC Compiler”를 선택합니다.
3. 탭에서 “Toolchain executables”를 누릅니다.
4. 아래 탭에서 “Additional Paths”를 누릅니다.
5. “Add” 버튼을 눌러 디렉토리를 추가해 줍니다.

3.2.3. Compiler installation directory (Linux)



위 그림은 리눅스 사용자들을 위한 크로스 컴파일러에 관한 설정입니다. 크로스 컴파일러는 GPH SDK에 포함되어 있는 GCC Cross Compiler를 사용합니다. 크로스 컴파일러가 설치되어 있는 디렉토리를 지정해주고, 실행파일들을 지정해 줍니다.

선택설정을 변경하려면,

1. "Settings" 메뉴에서 "Compiler and Debugger" 리스트를 선택합니다.
2. "Selected compiler"리스트에서 "GNU ARM GCC Compiler"를 선택합니다.
3. 탭에서 "Toolchain executables"를 누릅니다.
4. "Compiler"s installation directory"를 컴파일러가 설치된 디렉토리로 변경합니다.
5. 아래 탭에서 "Program Files"를 누릅니다.
6. 탐색 버튼을 눌러서, 해당 실행 파일을 선택합니다.

4. New Project

4.1. Creating Project

1) For Window OS

새로운 프로젝트를 생성할 때 사용되는 템플릿은 Codeblocks 설치시에 자동으로 해당 디렉토리에 생성됩니다.

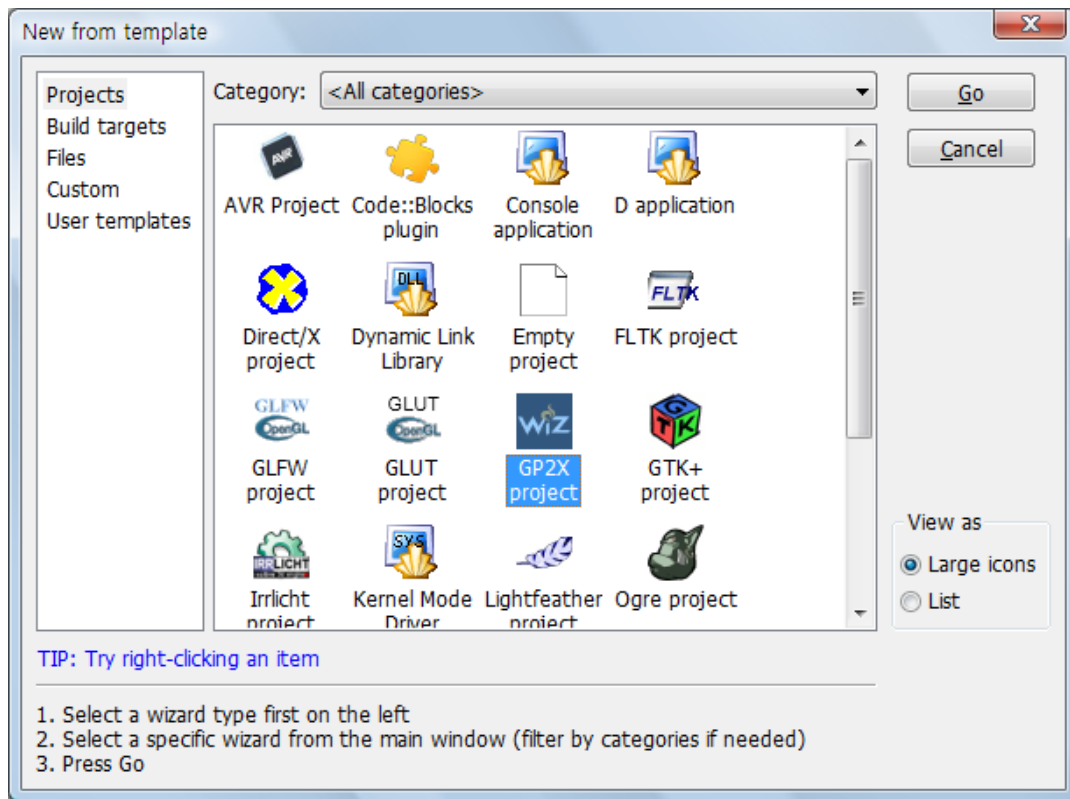
2) For Linux OS

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
```

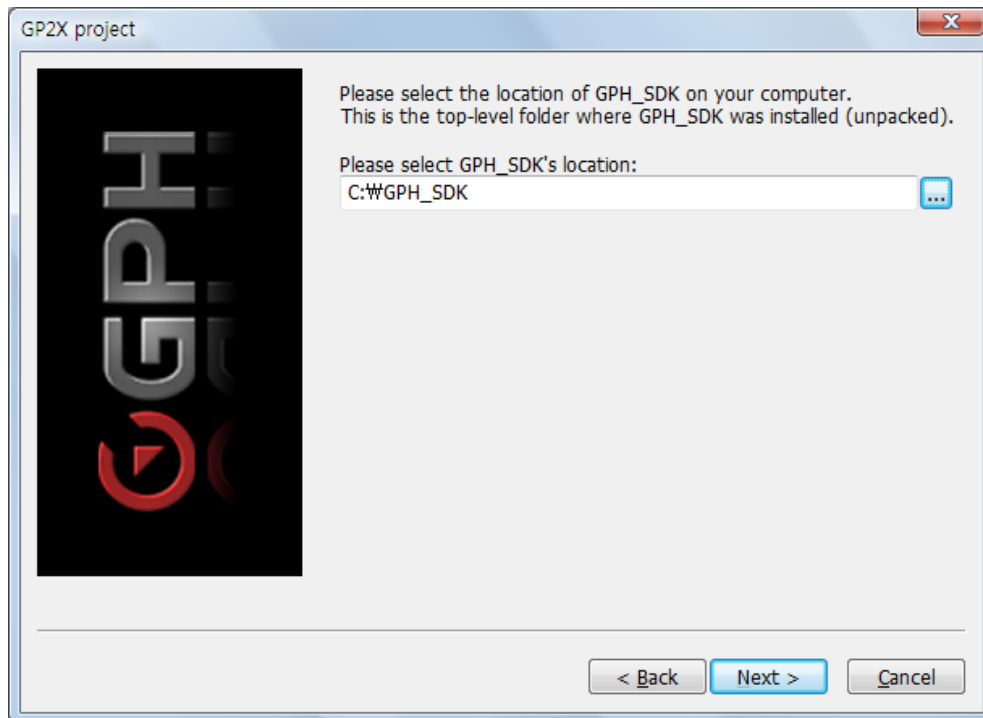
4.2. Using Templates



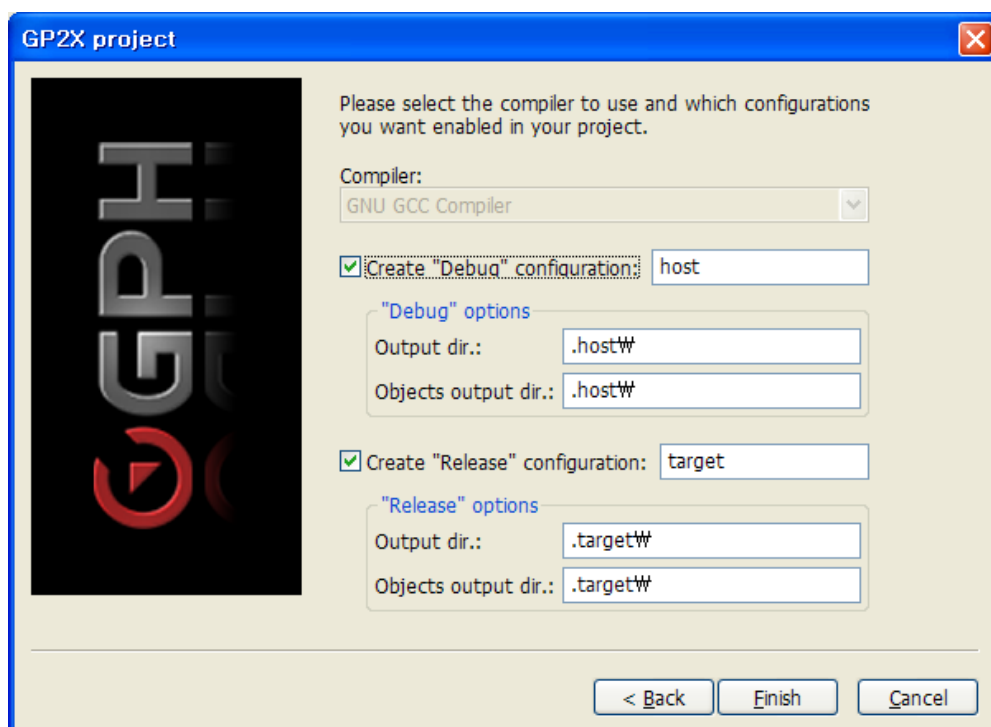
GPH SDK 는 편리하게 새로운 프로젝트를 생성할 수 있도록 템플릿을 포함하고 있습니다. 템플릿은 게임개발용 및 어플리케이션개발용으로 선택할 수 있습니다. 템플릿은 GPH SDK 를 설치할 때 자동으로 설치됩니다. 템플릿을 수동으로 설치하고자 하는 경우에는 GPH_SDK/tools/codeblocks 디렉토리를 참고하시기 바랍니다.

템플릿을 사용하여 프로젝트를 생성하려면,

1. “File” 메뉴에서 “New” -> “Project” 리스트를 선택합니다.
2. 왼쪽 리스트에서 “Projects”를 선택합니다.
3. “GP2X project” 아이콘을 누릅니다.
4. 해당 위저드의 안내에 따라 프로젝트를 생성합니다.



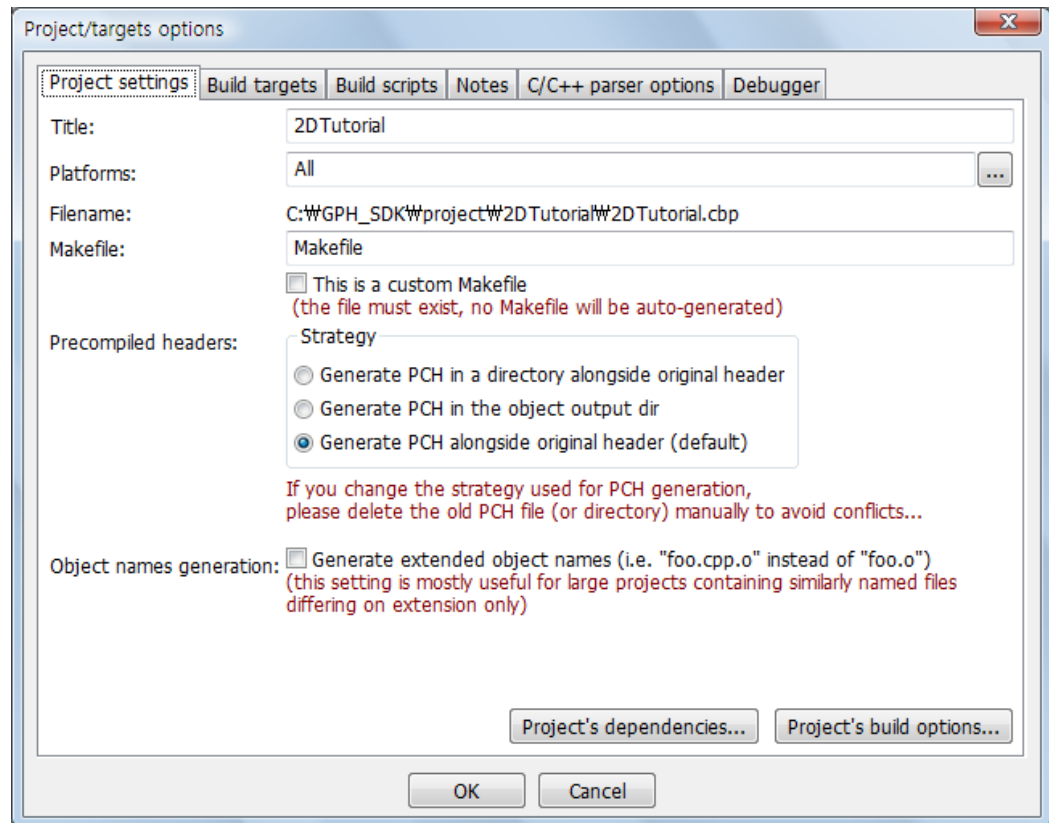
위 그림과 같이 GPH SDK 가 설치된 디렉토리를 지정해 줍니다.



위 그림과 같이 host, target 컴파일 옵션을 지정해 줍니다.

5. Project/target options

5.1. Project settings



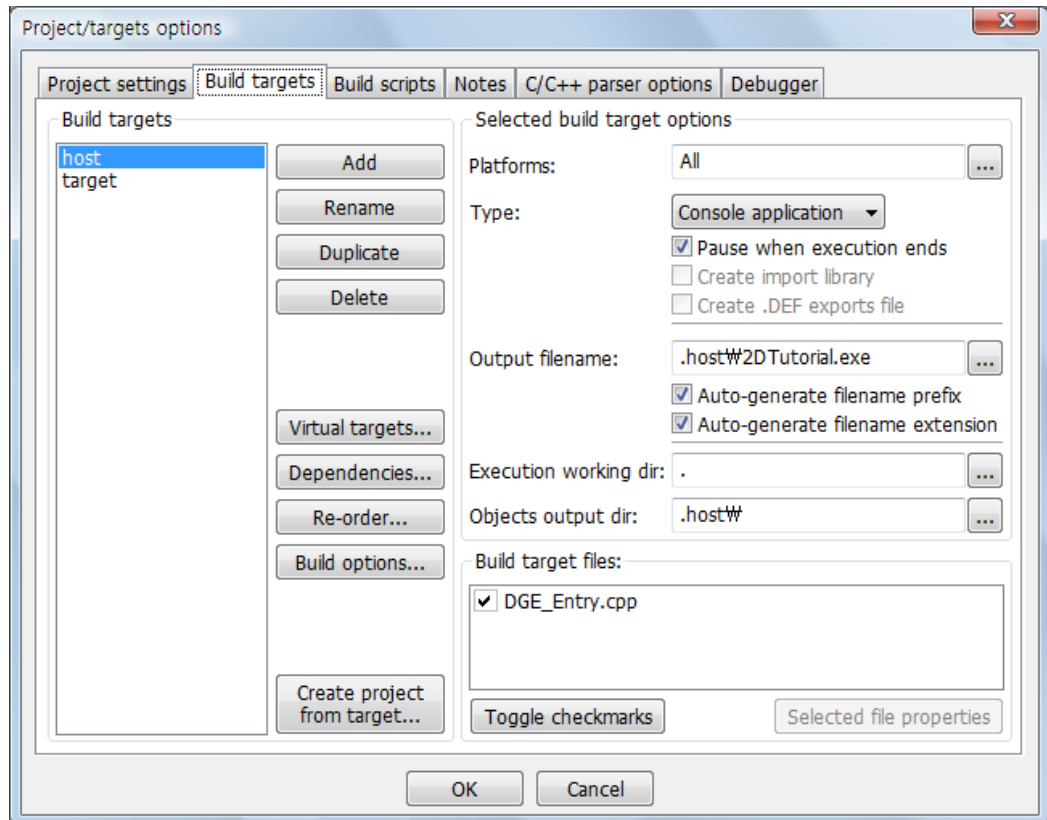
위 그림은 프로젝트의 속성을 설정하는 화면입니다. 외부의 Makefile 을 사용하는 경우 “This is custom Makefile”을 선택합니다.

선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. “Project” 메뉴에서 “Properties” 리스트를 선택합니다.
3. 탭에서 “Project settings”를 누릅니다.

5.2. Build targets options

5.2.1. Build targets -> host

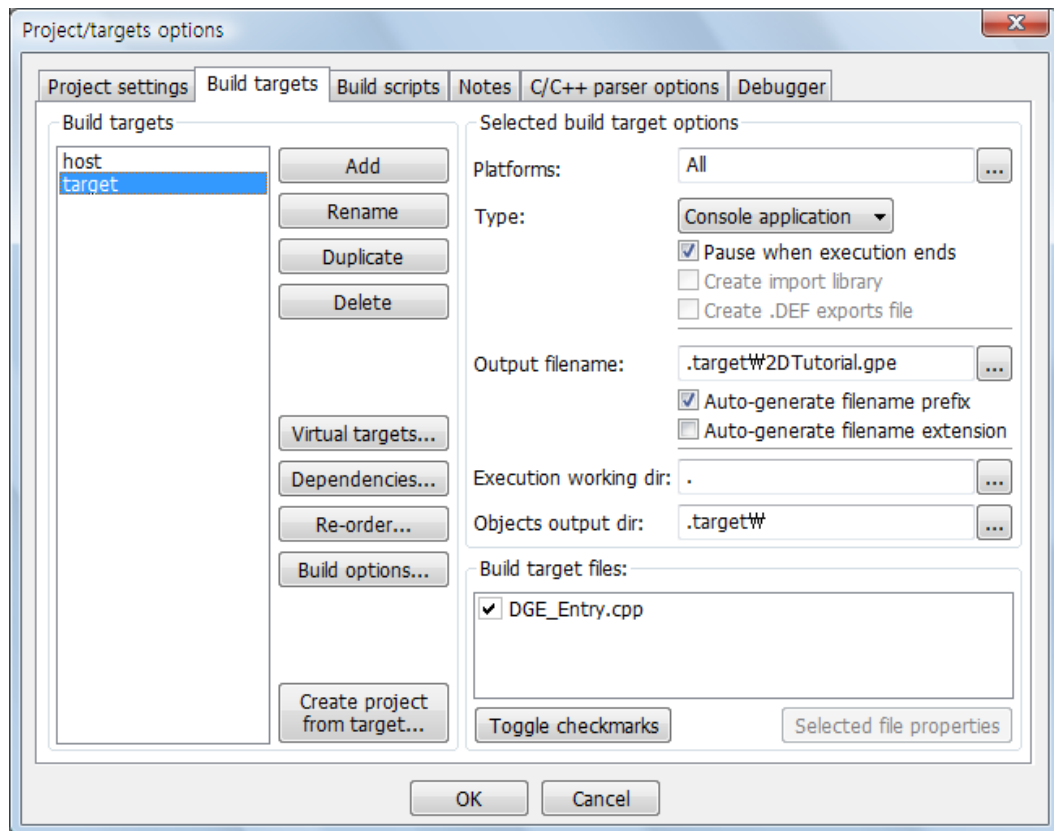


위 그림은 호스트 컴파일시의 빌드 타겟을 설정하는 화면입니다. 오브젝트 파일과 실행파일은 ".host" 디렉토리에 생성되며 필요한 경우 다른 디렉토리로 변경할 수 있습니다. 실행파일은 ".exe" 의 확장자로 생성되도록 설정되었습니다.

선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. "Project" 메뉴에서 "Properties" 리스트를 선택합니다.
3. 탭에서 "Build targets"를 누릅니다.
4. "Build targets" 리스트에서 "host"를 선택합니다.

5.2.2. Build targets -> target



위 그림은 크로스 컴파일시의 빌드 타겟을 설정하는 화면입니다. 오브젝트 파일과 실행파일은 ".target" 디렉토리에 생성되며 필요한 경우 다른 디렉토리로 변경할 수 있습니다. 실행파일은 ".gpe"의 확장자로 생성되도록 설정되었습니다.

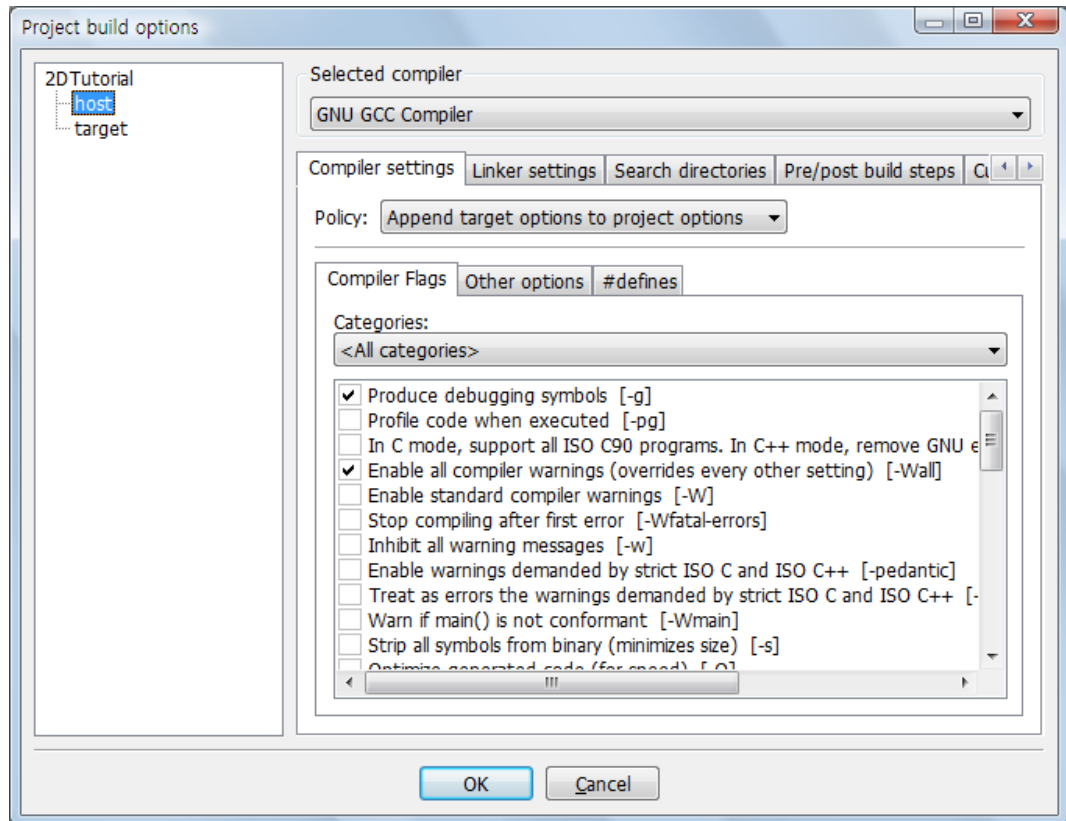
선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. "Project" 메뉴에서 "Properties" 리스트를 선택합니다.
3. 탭에서 "Build targets"를 누릅니다.
4. "Build targets" 리스트에서 "target"를 선택합니다.

5.3. Project build options

5.3.1. Project build options (host)

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

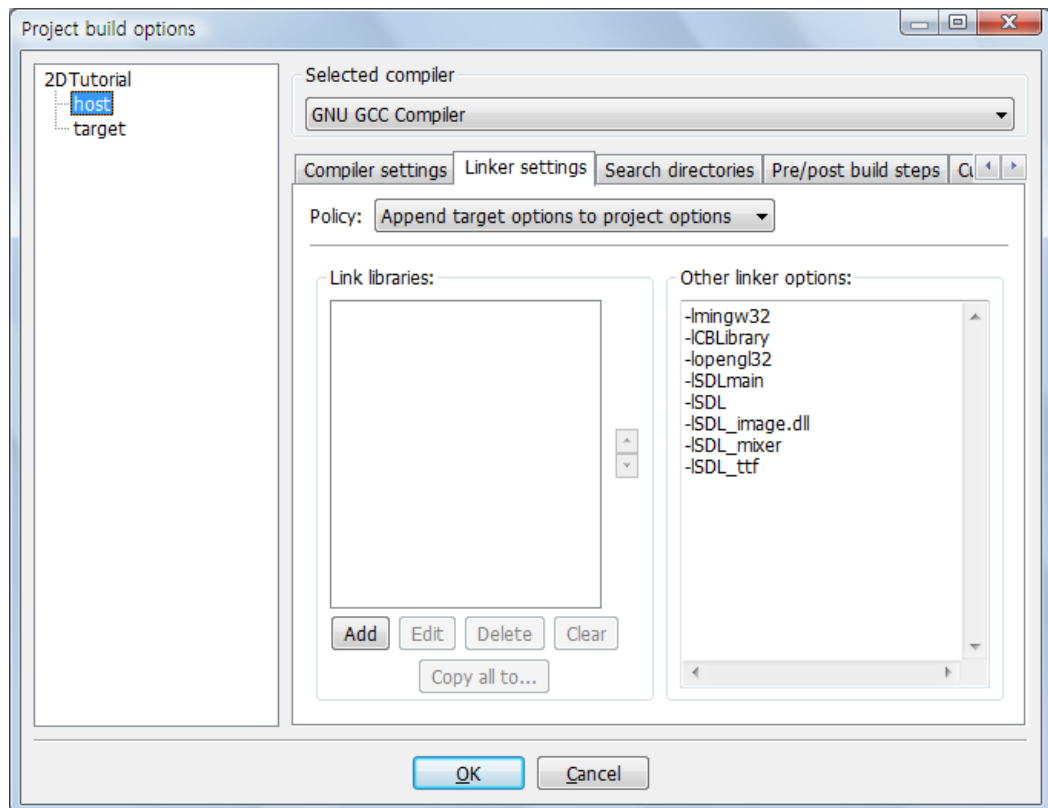


위 그림은 호스트 컴파일러의 컴파일 옵션 중에서 "Compiler Flags"를 설정하는 화면입니다.

선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. "Project" 메뉴에서 "Build options" 리스트를 선택합니다.
3. 왼쪽 리스트에서 "host"를 선택합니다.
4. "Selected compiler" 리스트에서 "GNU GCC Compiler"를 선택합니다.
5. 탭에서 "Compiler settings"를 누릅니다.
6. 아래탭에서 "Compiler flags"를 누릅니다.

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

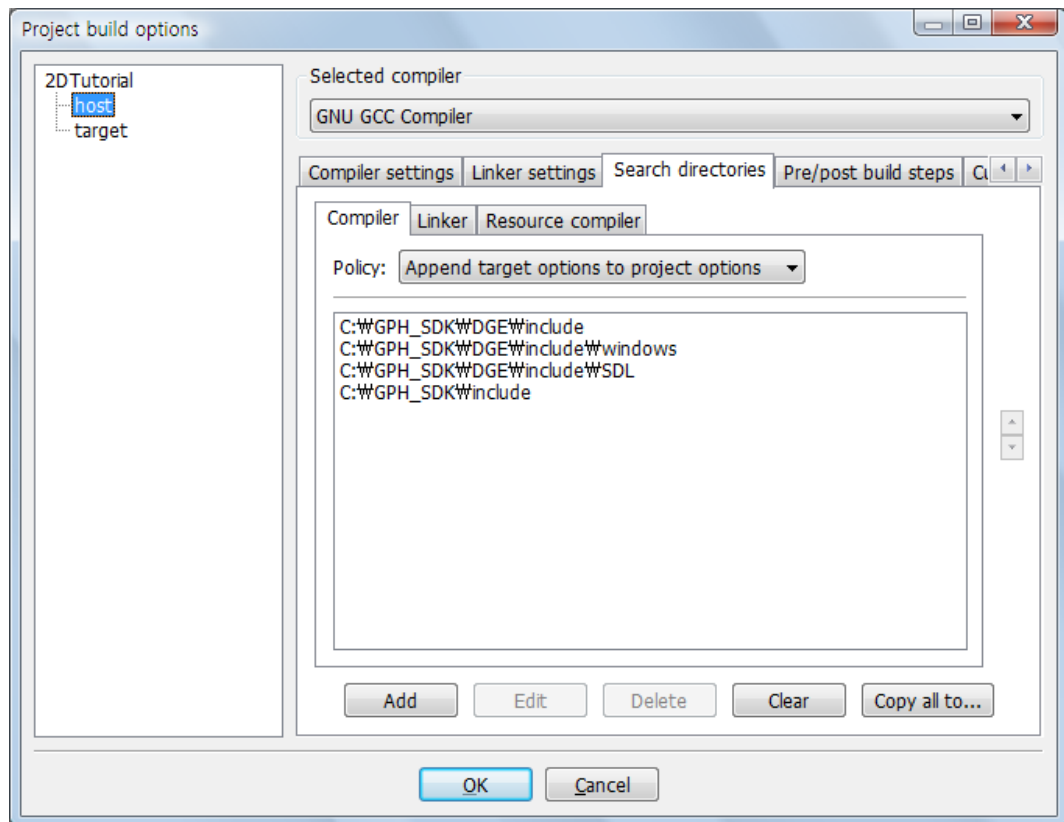


위 그림은 호스트 컴파일러의 링커옵션 중에서 라이브러리를 설정하는 화면입니다.

선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. "Project" 메뉴에서 "Build options" 리스트를 선택합니다.
3. 왼쪽 리스트에서 "host"를 선택합니다.
4. "Selected compiler" 리스트에서 "GNU GCC Compiler"를 선택합니다.
5. 탭에서 "Linker settings"를 누릅니다.
6. "Add" 버튼을 눌러 라이브러리를 추가할 수도 있고, 링커옵션으로 직접 입력할 수도 있습니다.

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

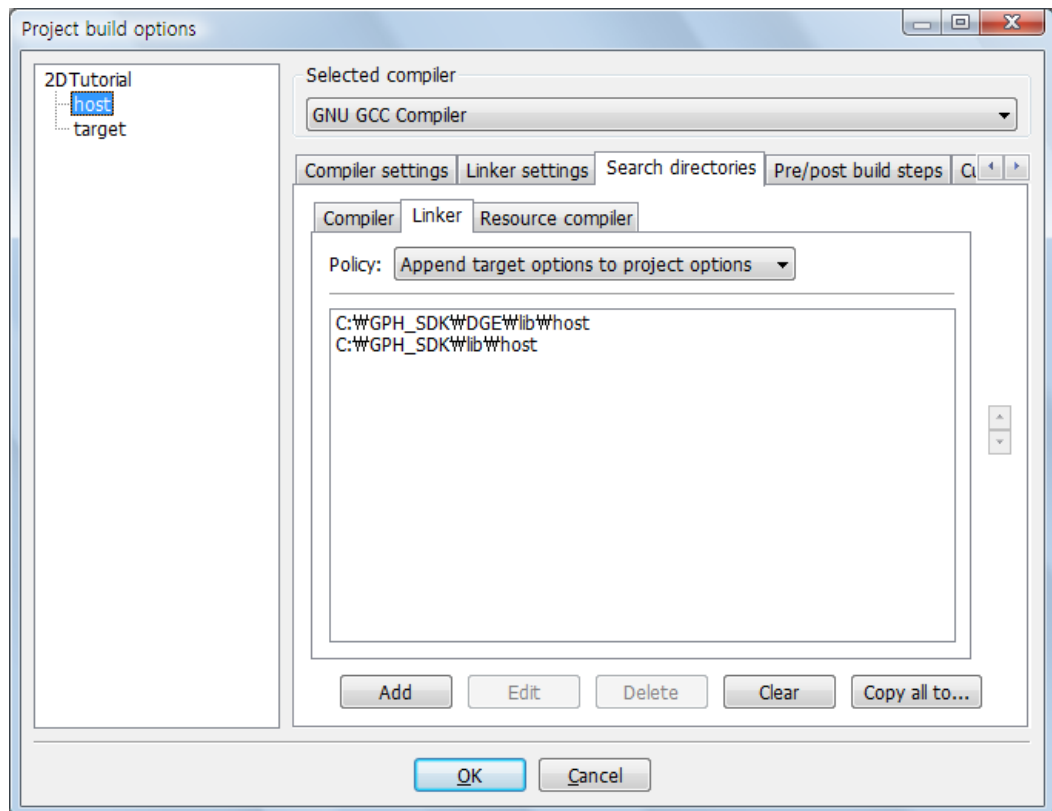


위 그림은 호스트 컴파일러의 컴파일옵션 중에서 헤더파일의 경로를 설정하는 화면입니다.

선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. "Project" 메뉴에서 "Build options" 리스트를 선택합니다.
3. 왼쪽 리스트에서 "host"를 선택합니다.
4. "Selected compiler" 리스트에서 "GNU GCC Compiler"를 선택합니다.
5. 탭에서 "Search directories"를 누릅니다.
6. 아래 탭에서 "Compiler"를 누릅니다.
7. "Add" 버튼을 눌러 헤더파일의 경로를 추가할 수 있습니다.

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



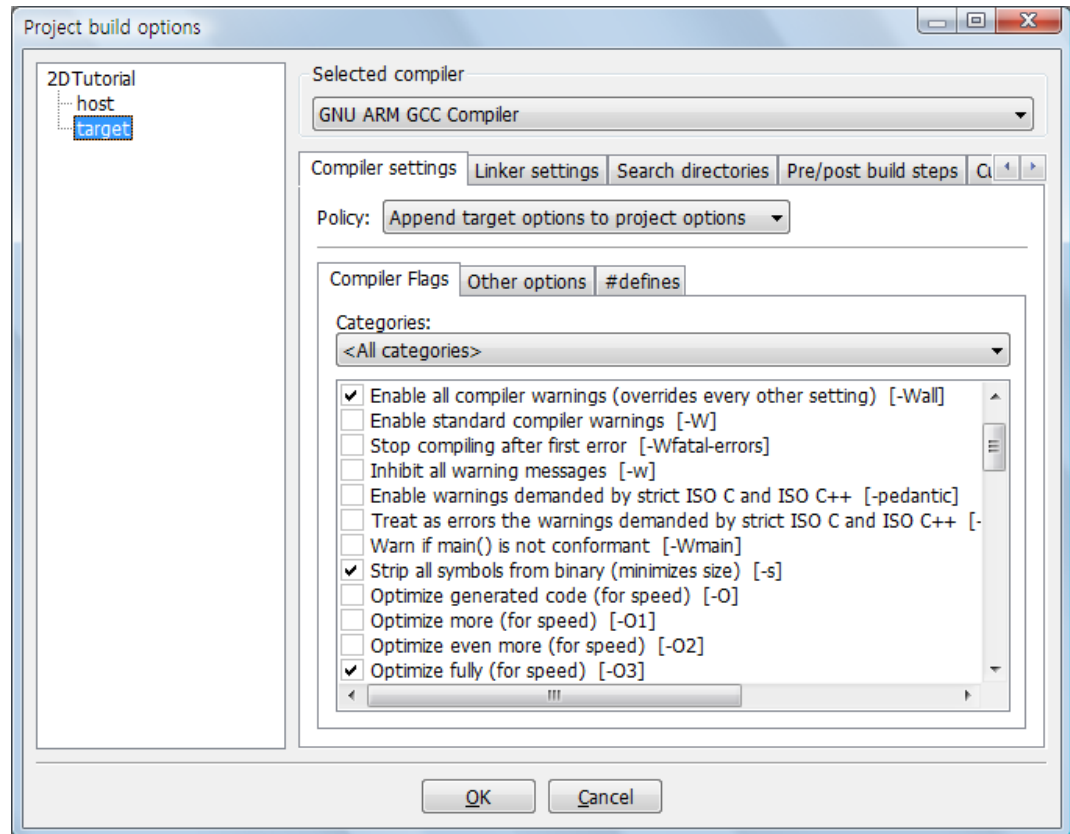
위 그림은 호스트 컴파일러의 링커옵션 중에서 라이브러리의 경로를 설정하는 화면입니다.

선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. "Project" 메뉴에서 "Build options" 리스트를 선택합니다.
3. 왼쪽 리스트에서 "host"를 선택합니다.
4. "Selected compiler" 리스트에서 "GNU GCC Compiler"를 선택합니다.
5. 탭에서 "Search directories"를 누릅니다.
6. 아래 탭에서 "Linker"를 누릅니다.
7. "Add" 버튼을 눌러 라이브러리의 경로를 추가할 수 있습니다.

5.3.2. Project build options (target)

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

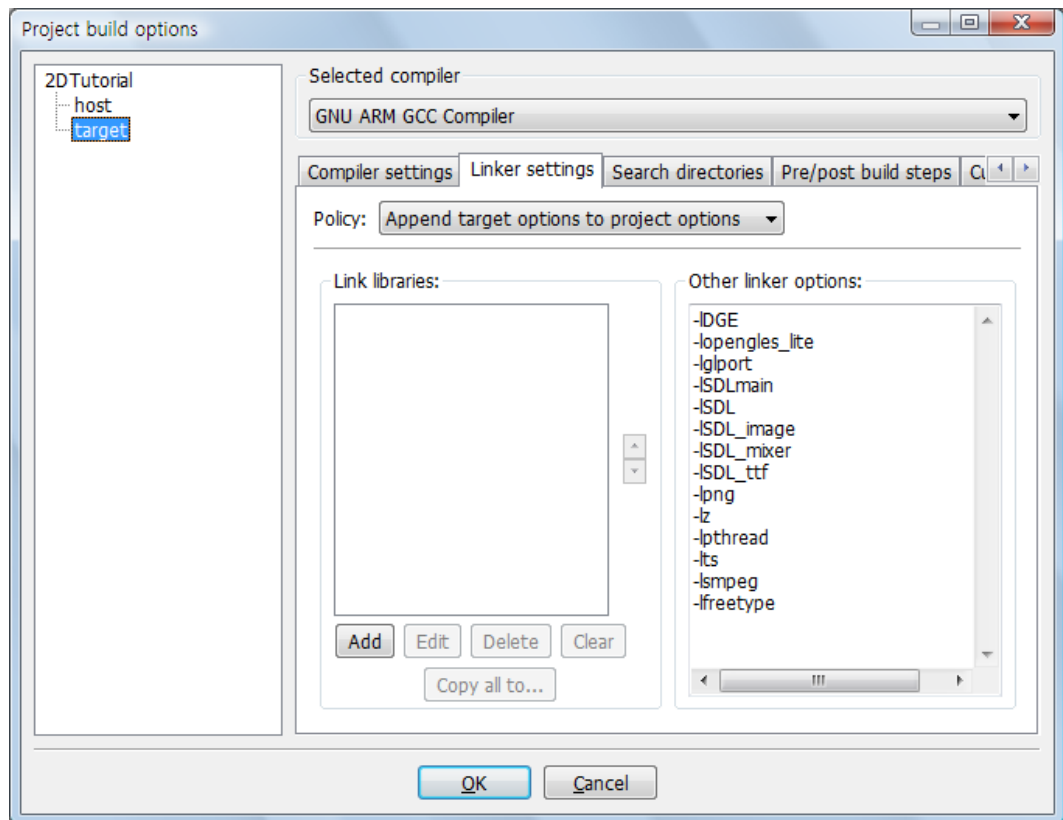


위 그림은 크로스 컴파일러의 컴파일 옵션 중에서 "Compiler Flags"를 설정하는 화면입니다.

선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. "Project" 메뉴에서 "Build options" 리스트를 선택합니다.
3. 왼쪽 리스트에서 "target"를 선택합니다.
4. "Selected compiler" 리스트에서 "GNU ARM GCC Compiler"를 선택합니다.
5. 탭에서 "Compiler settings"를 누릅니다.
6. 아래탭에서 "Compiler flags"를 누릅니다.

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

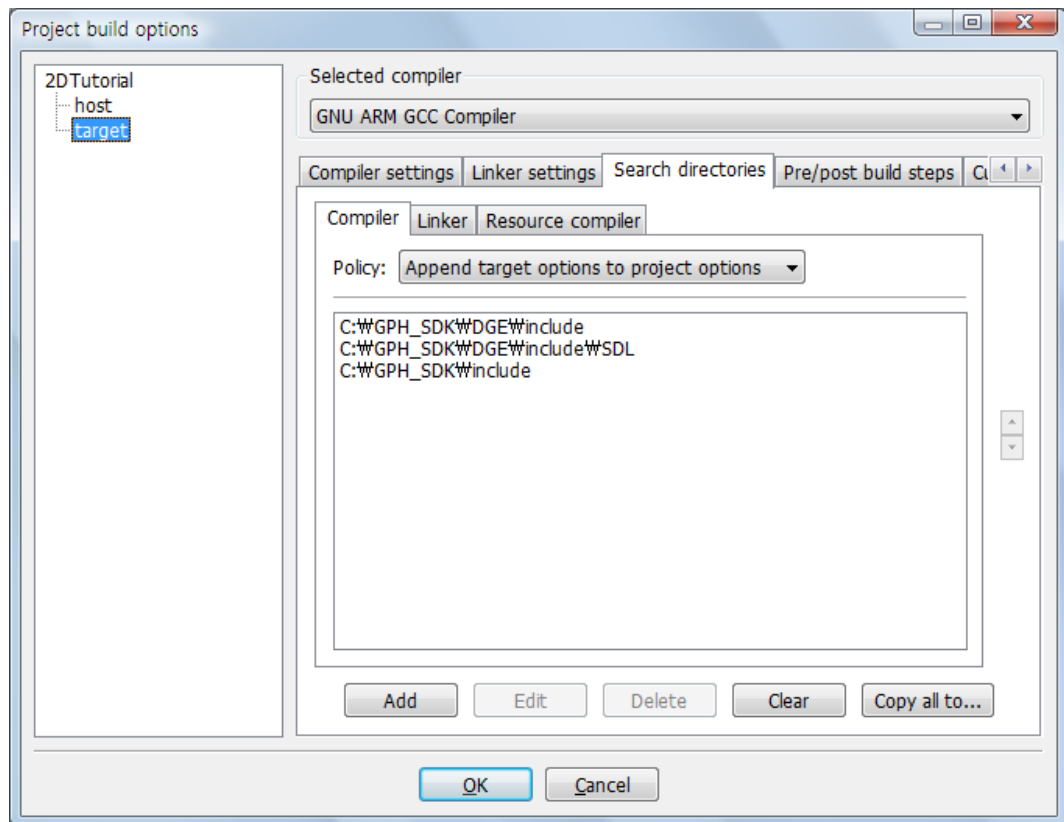


위 그림은 크로스 컴파일러의 링커옵션 중에서 라이브러리를 설정하는 화면입니다.

선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. "Project" 메뉴에서 "Build options" 리스트를 선택합니다.
3. 왼쪽 리스트에서 "target"를 선택합니다.
4. "Selected compiler" 리스트에서 "GNU ARM GCC Compiler"를 선택합니다.
5. 탭에서 "Linker settings"를 누릅니다.
6. "Add" 버튼을 눌러 라이브러리를 추가할 수도 있고, 링커옵션으로 직접 입력할 수도 있습니다.

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

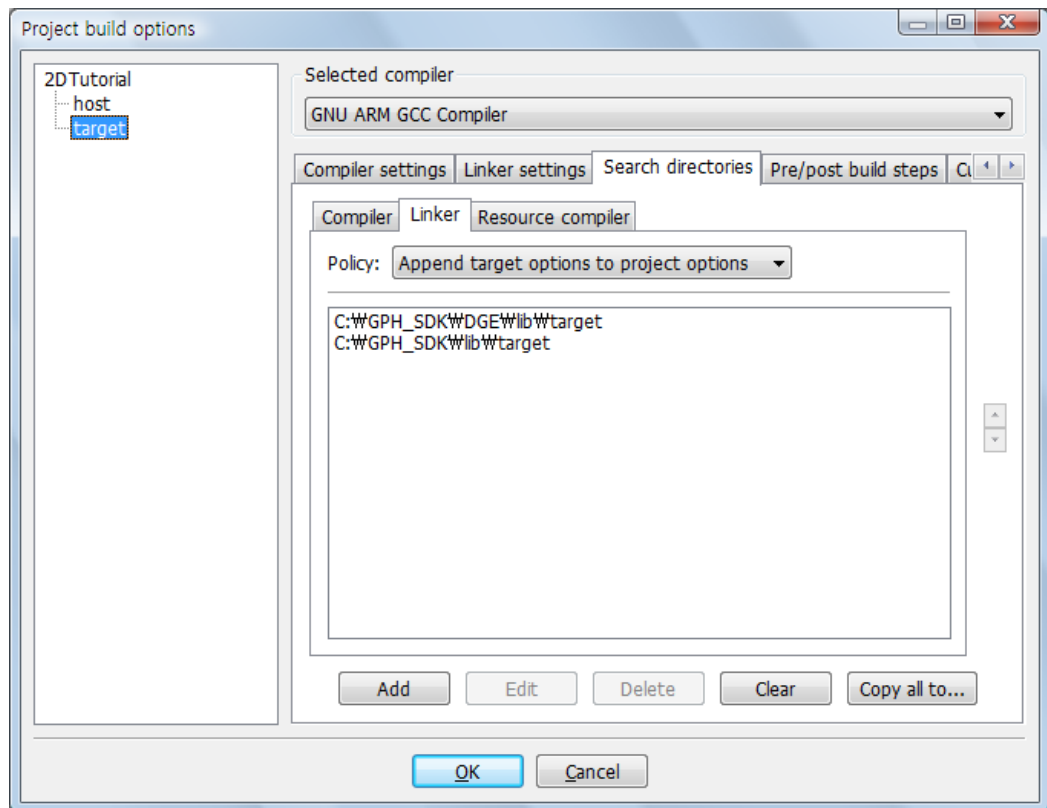


위 그림은 크로스 컴파일러의 컴파일옵션 중에서 헤더파일의 경로를 설정하는 화면입니다.

선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. "Project" 메뉴에서 "Build options" 리스트를 선택합니다.
3. 왼쪽 리스트에서 "target"를 선택합니다.
4. "Selected compiler" 리스트에서 "GNU ARM GCC Compiler"를 선택합니다.
5. 탭에서 "Search directories"를 누릅니다.
6. 아래 탭에서 "Compiler"를 누릅니다.
7. "Add" 버튼을 눌러 헤더파일의 경로를 추가할 수 있습니다.

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

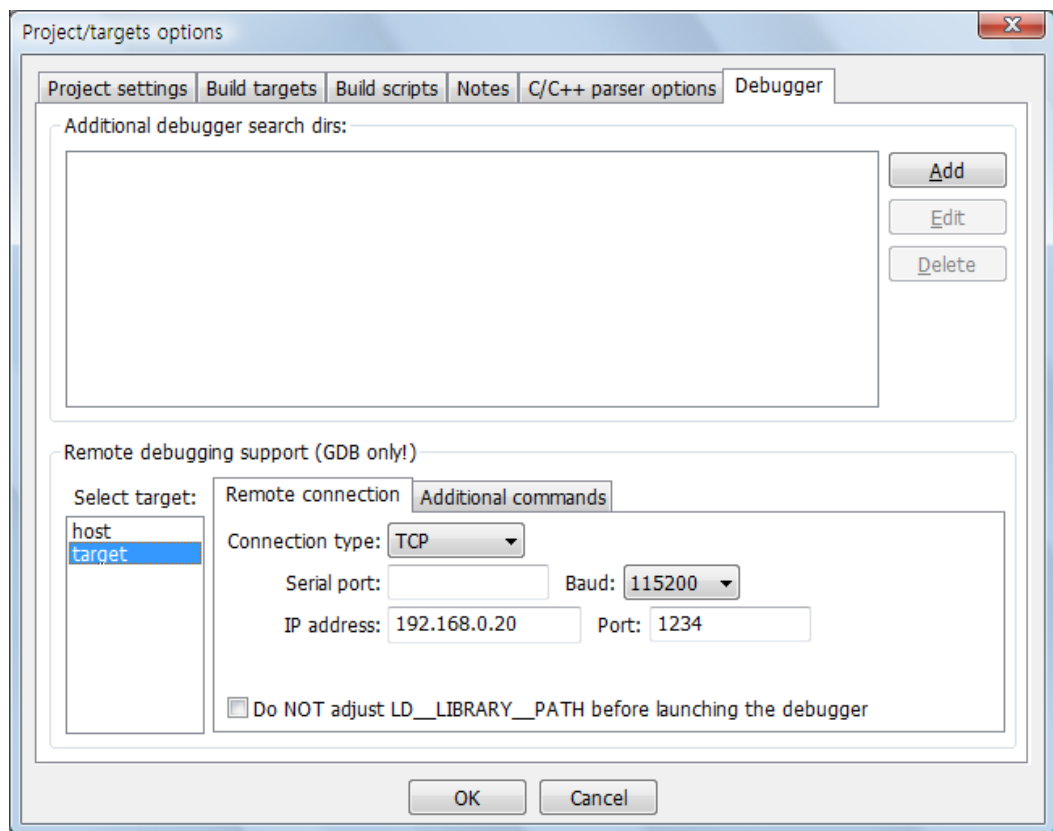


위 그림은 크로스 컴파일러의 링커옵션 중에서 라이브러리의 경로를 설정하는 화면입니다.

선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. "Project" 메뉴에서 "Build options" 리스트를 선택합니다.
3. 왼쪽 리스트에서 "target"를 선택합니다.
4. "Selected compiler" 리스트에서 "GNU ARM GCC Compiler"를 선택합니다.
5. 탭에서 "Search directories"를 누릅니다.
6. 아래 탭에서 "Linker"를 누릅니다.
7. "Add" 버튼을 눌러 라이브러리의 경로를 추가할 수 있습니다.

5.4. Debugger



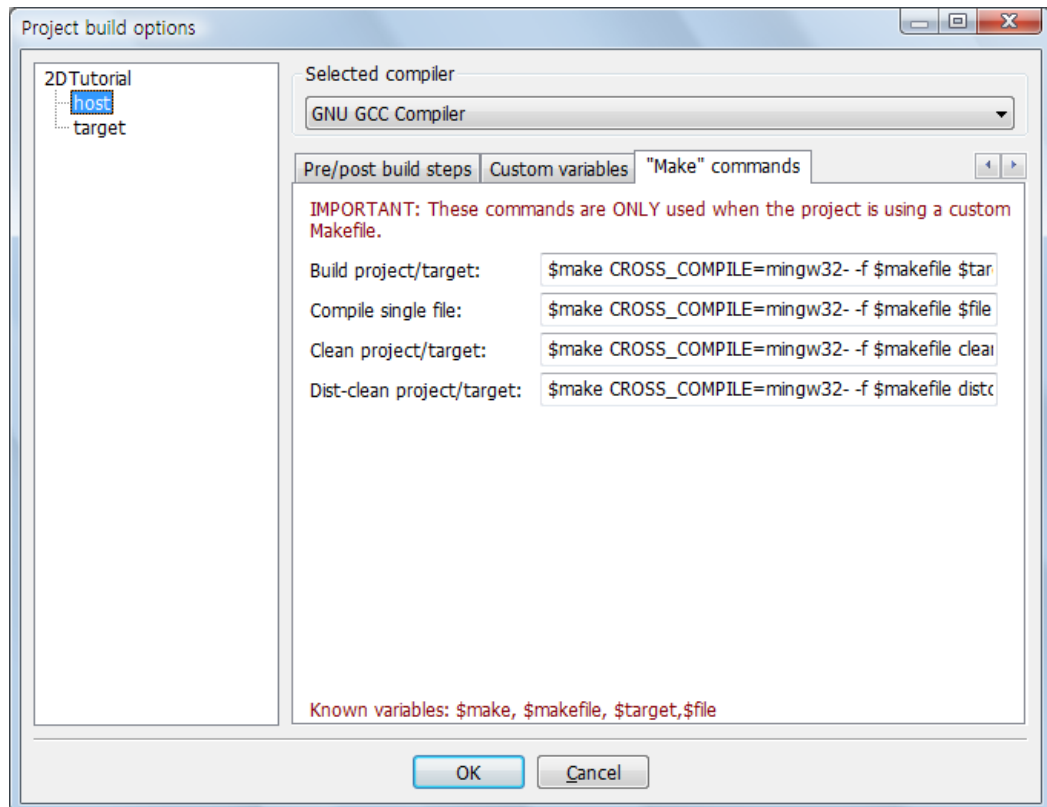
위 그림은 원격 디버깅을 하기 위한 설정을 보여주는 화면이다. 호스트 머신과 타겟 머신이 네트워크로 연결되어 있는 경우에만 사용할 수 있다. Wifi 커널 모듈과 네트워크 연결을 위한 툴은 GPH_SDK/tools 디렉토리에 있다. gdbserver가 실행되고 있는 타겟머신의 IP address와 port number를 입력할 수 있다.

선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. "Project" 메뉴에서 "Properties" 리스트를 선택합니다.
3. 탭에서 "Debugger"를 누릅니다.
4. "Select target" 리스트에서 target을 선택합니다.

6. Using Makefile

6.1. Host "Make" commands

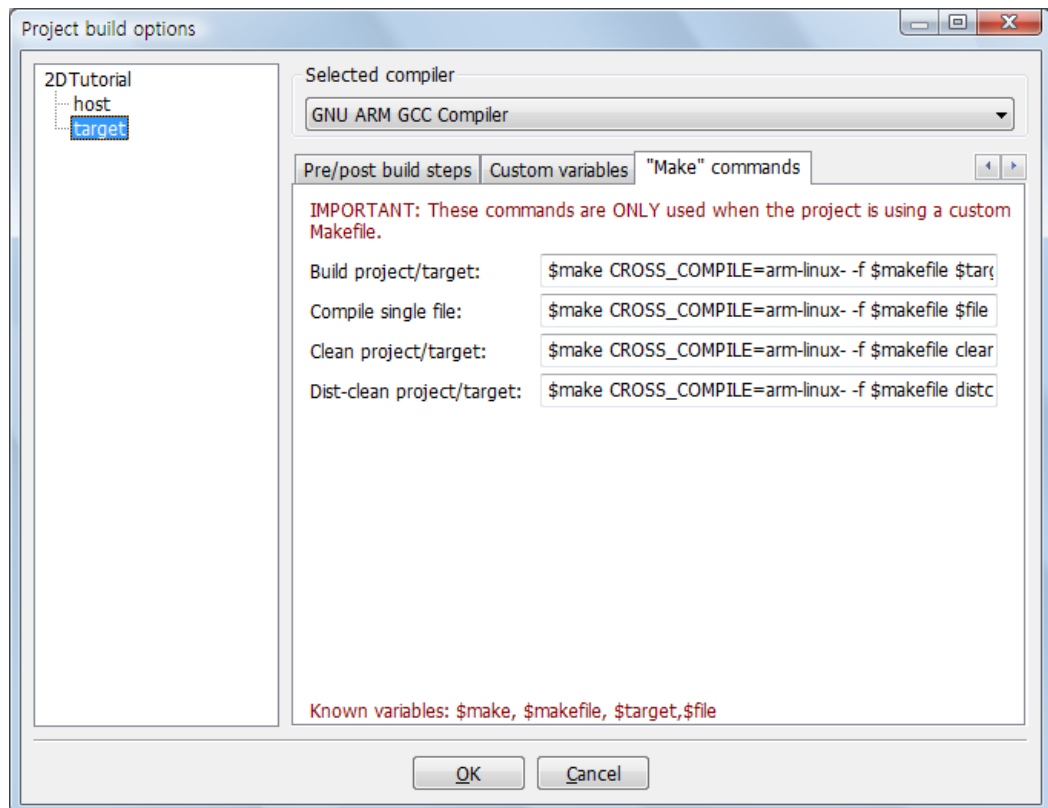


위 그림은 외부의 Makefile 을 사용하는 경우의 호스트 컴파일에 관한 설정입니다. Makefile을 사용하여 컴파일을 하려면, 5.1절 Project settings를 참고하시기 바랍니다. CROSS_COMPILE 환경변수를 지정하여 Makefile 에서 사용할 수 있도록 해 줍니다.

선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. "Project" 메뉴에서 "Build options" 리스트를 선택합니다.
3. 왼쪽 리스트에서 "host"를 선택합니다.
4. "Selected compiler" 리스트에서 "GNU GCC Compiler"를 선택합니다.
5. 탭에서 "Make commands"를 누릅니다.

6.2. Target "Make" commands



위 그림은 외부의 Makefile 을 사용하는 경우의 크로스 컴파일에 관한 설정입니다. Makefile을 사용하여 컴파일을 하려면, 5.1절 Project settings를 참고하시기 바랍니다. CROSS_COMPILE 환경변수를 지정하여 Makefile 에서 사용할 수 있도록 해 줍니다.

선택설정을 변경하려면,

1. 프로젝트를 오픈하거나, 새로운 프로젝트를 생성합니다.
2. "Project" 메뉴에서 "Build options" 리스트를 선택합니다.
3. 왼쪽 리스트에서 "target"를 선택합니다.
4. "Selected compiler" 리스트에서 "GNU GCC Compiler"를 선택합니다.
5. 탭에서 "Make commands"를 누릅니다.

6.3. Makefile : example

Makefile은 호스트 컴파일 및 크로스 컴파일에 모두 사용될 수 있도록 구성되어 있습니다. 소스, 헤더파일 그리고 라이브러리등이 추가되는 경우에는 편집기를 이용하여 추가할 수 있습니다.

```

TARGET    = 2DTutorial
OBJ        = 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): $(OBJ)
    $(CPP) $(OBJ) -o $@ $(LIBS) $(LFLAGS)

all host target: $(BIN)

clean cleanhost cleantarget:

```