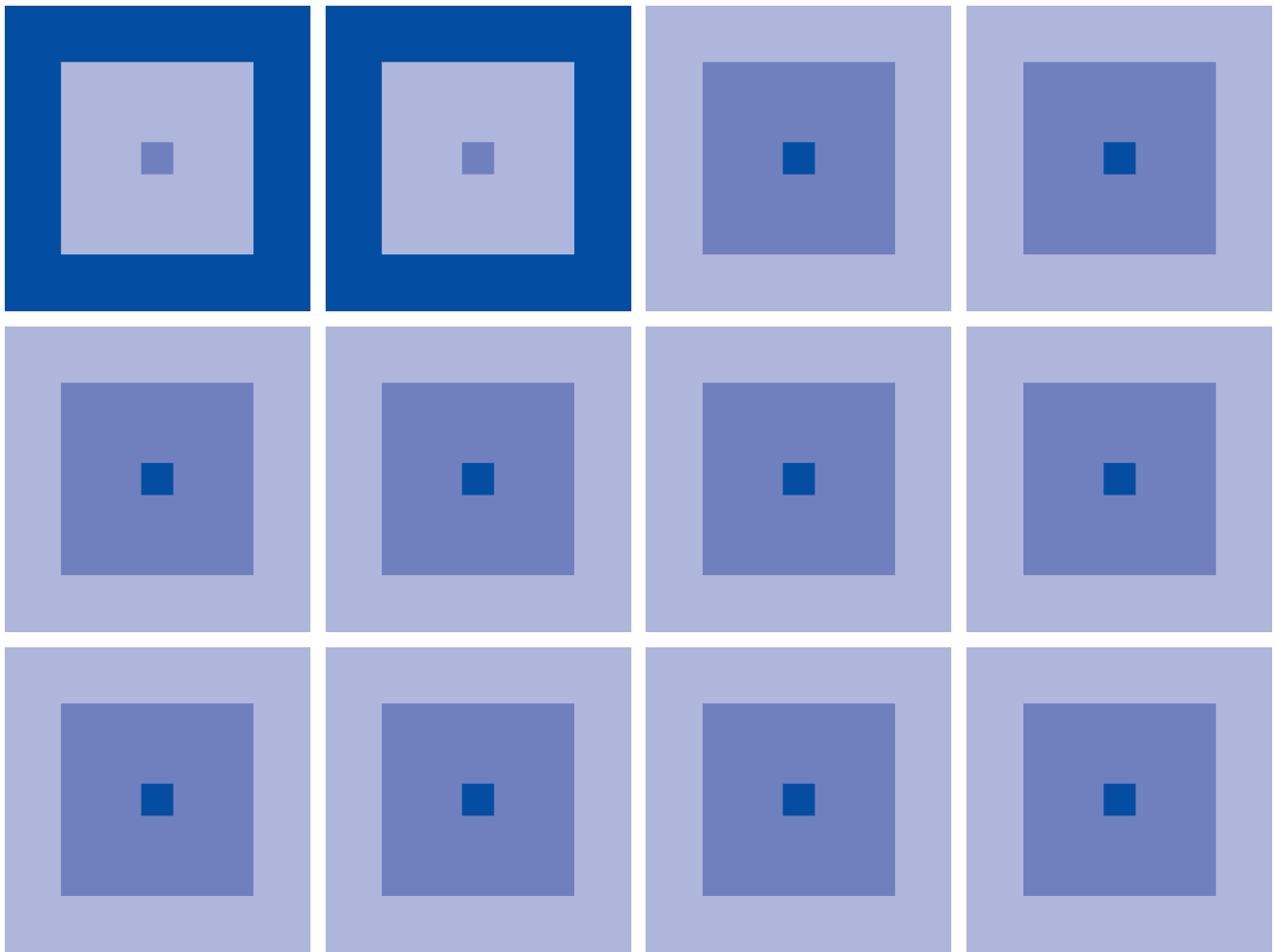


CMOS 16-BIT SINGLE CHIP MICROCOMPUTER  
**S5U1C17001C** Manual  
(C Compiler Package for S1C17 Family)  
(Ver. 1.1)



## ***NOTICE***

---

*No part of this material may be reproduced or duplicated in any form or by any means without the written permission of Seiko Epson. Seiko Epson reserves the right to make changes to this material without notice. Seiko Epson does not assume any liability of any kind arising out of any inaccuracies contained in this material or due to its application or use in any product or circuit and, further, there is no representation that this material is applicable to products requiring high level reliability, such as medical products. Moreover, no license to any intellectual property rights is granted by implication or otherwise, and there is no representation or warranty that anything made in accordance with this material will be free from any patent or copyright infringement of a third party. This material or portions thereof may contain technology or the subject relating to strategic products under the control of the Foreign Exchange and Foreign Trade Law of Japan and may require an export license from the Ministry of Economy, Trade and Industry or other approval from another government agency.*

Windows 2000 and Windows XP are registered trademarks of Microsoft Corporation, U.S.A.

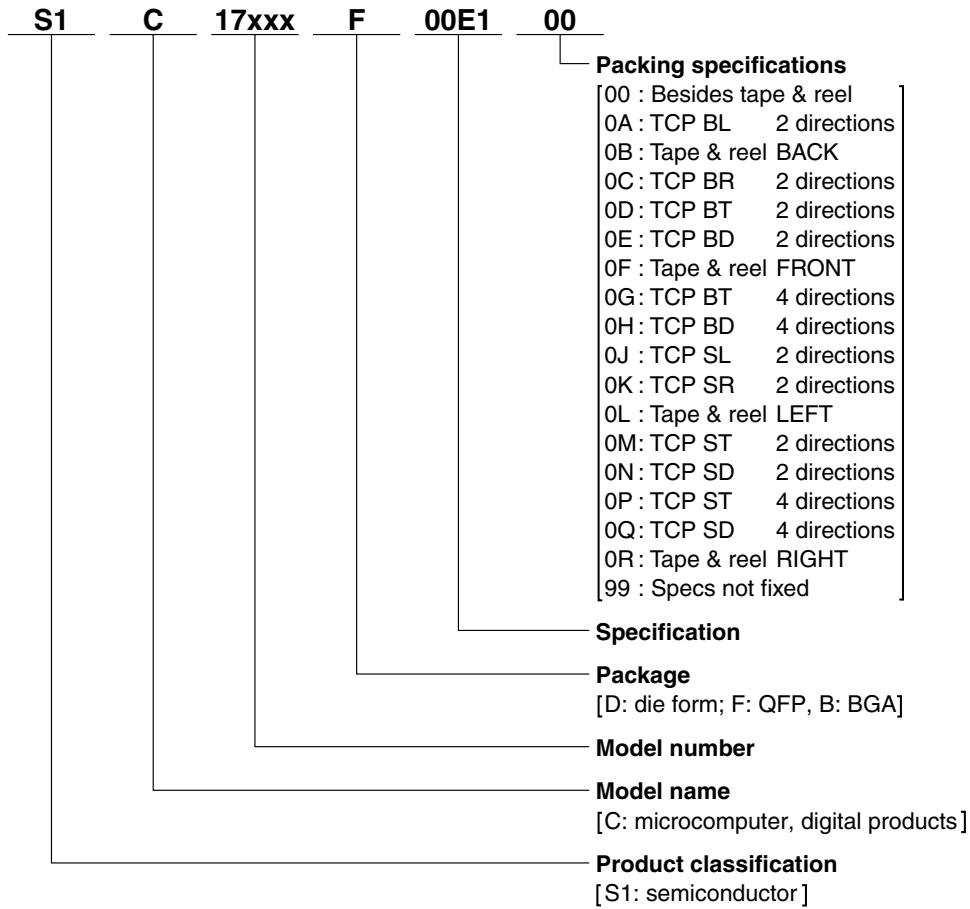
PC/AT and IBM are registered trademarks of International Business Machines Corporation, U.S.A.

All other product names mentioned herein are trademarks and/or registered trademarks of their respective owners.

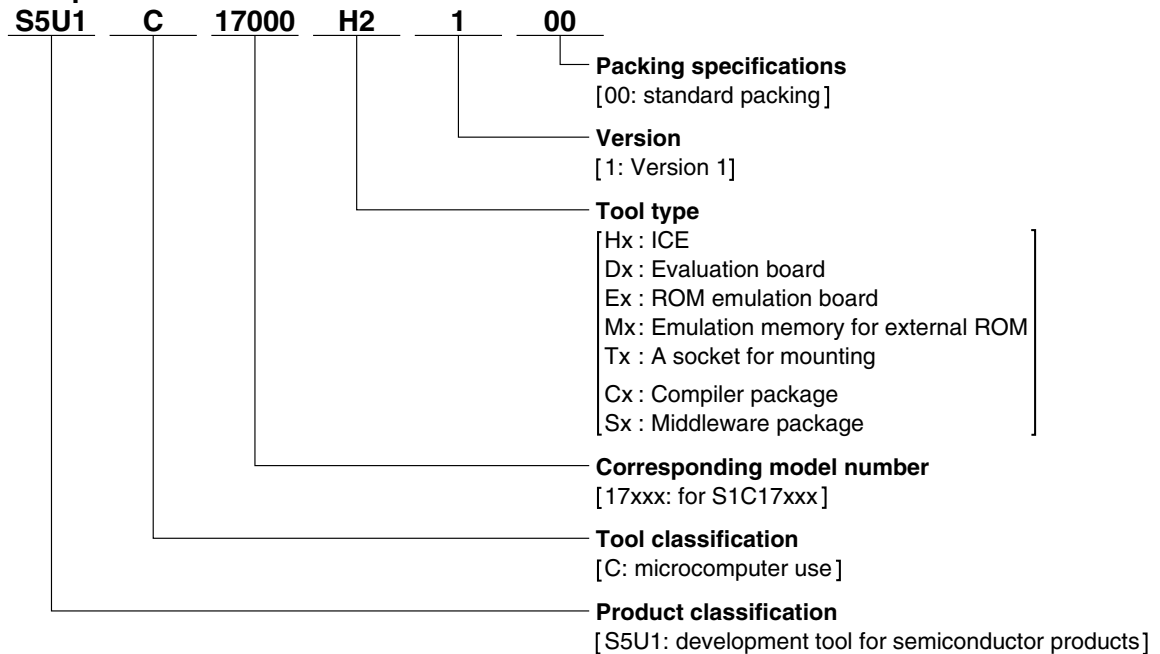
© **SEIKO EPSON CORPORATION** 2007, All rights reserved.

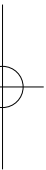
## Configuration of product number

### Devices



### Development tools





## Introduction

This document describes the development procedure from compiling C source files to debugging and creating the mask data which is finally submitted to Seiko Epson. It also explains how to use each development tool of the S1C17 Family C Compiler Package common to all the models of the S1C17 Family.

## How To Read the Manual

This manual was edited particularly for those who are engaged in program development. Therefore, it assumes that the reader already possesses the following fundamental knowledge:

- Knowledge about C language (based on ANSI C) and C source creation methods
- Knowledge about the gnu C, binutils, gnu make and the linker script for the gnu linker (ld)
- Basic knowledge about assembler language
- Basic knowledge about the general concept of program development by a C compiler and an assembler
- Basic operating methods for Windows 2000 or Windows XP.

Please refer to manuals or general documents which describe ANSI C, gnu tools and Windows, for the above contents.

### Before installation

See Chapter 1. Chapter 1 describes the composition of this package, and provides a general outline of each tool.

### Installation

Install the tools following the installation procedure described in Chapter 2.

### To understand the flow of program development and the operating procedure

See the Tutorial described in Chapter 3. This will give you an overview of program development using the C compiler to the debugger and how to make the mask data.

### For coding

See the necessary parts in Chapter 4. Chapter 4 describes notes on creating source files and the grammar for the assembler language. Also refer to the following manuals when coding:

S1C17xxx Technical Manual

Covers device specifications, and the operation and control method of the peripheral circuits.

S1C17 Core Manual

Has the instructions and details the functions and operation of the Core CPU.

### For debugging

Chapter 10 explains details of the debugger. Sections 10.1 to 10.6 give an overview of the functions of the debugger. See Section 10.7 for details of the debug commands. Also refer to the following manuals to understand operations of the debugging tools:

S1C17 Family In-Circuit Debugger Manual

Explains the functions and handling methods of the ICD Mini (S5U1C17001H).

### For details of each tool

Refer to Chapters 5 to 11 and gnu tool manuals for details.

## Manual Notations

This manual was prepared by following the notation rules detailed below:

### Samples

The sample screens provided in the manual are all examples of displays under Windows 2000/XP. These displays may vary according to the system or fonts used.

### Names of each part

The names or designations of the windows, menus and menu commands, buttons, dialog boxes, and keys are annotated in brackets [ ]. Examples: [Command] window, [File] menu, [Stop] button, [q] key, etc.

### Names of instructions and commands

The CPU instructions and the debugger commands that can be written in either uppercase or lowercase characters are annotated in lowercase characters in this manual, except for user-specified symbols. A fixed-width font is used to describe these words.

### Notation of numeric values

Numeric values are described as follows:

Decimal numbers: Not accompanied by any prefix or suffix (e.g., 123, 1000).

Hexadecimal numbers: Accompanied by the prefix "0x" (e.g., 0x0110, 0xffff).

Binary numbers: Accompanied by the prefix "0b" (e.g., 0b0001, 0b10).

However, please note that some sample displays may indicate hexadecimal or binary numbers not accompanied by any symbol.

### Mouse operations

To click: The operation of pressing the left mouse button once, with the cursor (pointer) placed in the intended location, is expressed as "to click". The clicking operation of the right mouse button is expressed as "to right-click".

To double-click: Operations of pressing the left mouse button twice in a row, with the cursor (pointer) placed in the intended location, are all expressed as "to double-click".

To drag: The operation of clicking on a file (icon) with the left mouse button and holding it down while moving the icon to another location on the screen is expressed as "to drag".

To select: The operation of selecting a menu command by clicking is expressed as "to select".

### Key operations

The operation of pressing a specific key is expressed as "to enter a key" or "to press a key".

A combination of keys using "+", such as [Ctrl]+[C] keys, denotes the operation of pressing the [C] key while the [Ctrl] key is held down. Sample entries through the keyboard are not indicated in [ ].

In this manual, all the operations that can be executed with the mouse are described only as mouse operations.

For operating procedures executed through the keyboard, refer to the Windows manual or help screens.

### General forms of commands, startup options, and messages

Items given in [ ] are those to be selected by the user, and they will work without any key entry involved.

An annotation enclosed in < > indicates that a specific name should be placed here. For example, <file name> needs to be replaced with an actual file name.

### Development tool name

ICD: Indicates the ICD Mini (S5U1C17001H) or the ICD board.

## – Contents –

<b>1 General</b> .....	<b>1-1</b>
1.1 Features .....	1-1
1.2 Tool Composition.....	1-2
1.2.1 Composition of Package .....	1-2
1.2.2 Outline of Software Tools .....	1-2
<b>2 Installation</b> .....	<b>2-1</b>
2.1 Working Environment .....	2-1
2.2 Installation Method .....	2-2
<b>3 Software Development Procedures</b> .....	<b>3-1</b>
3.1 Software Development Flow .....	3-1
3.2 Software Development Using the IDE.....	3-3
3.3 Tutorial 1 (Basic IDE and Debugger Operations) .....	3-7
3.3.1 Starting the IDE .....	3-7
3.3.2 Creating a Project .....	3-9
3.3.3 Creating, Adding, and Editing a Source File.....	3-12
3.3.4 Editing the Build Options and the Linker Script .....	3-18
3.3.5 Building a Program .....	3-26
3.3.6 Debugging a Program.....	3-27
3.3.7 Creating ROM Data .....	3-41
3.4 Tutorial 2 (Using the User Makefiles) .....	3-43
3.4.1 Creating a Project.....	3-43
3.4.2 Importing Source Files.....	3-45
3.4.3 Disabling the GNU17 File Builder .....	3-47
3.4.4 Setting and Correcting the Makefile.....	3-48
3.4.5 Building a Project.....	3-49
3.4.6 Starting the Debugger.....	3-49
3.5 Tutorial 3 (Importing an IDE Project).....	3-51
3.6 Tutorial 4 (How to Use ES-Sim17) .....	3-54
3.6.1 Settings Required for Launching ES-Sim17 .....	3-54
3.6.2 How to Launch ES-Sim17 in the Existing Project .....	3-57
3.7 Debugging Environment.....	3-63
3.8 Sections and Linkage.....	3-64
<b>4 Source Files</b> .....	<b>4-1</b>
4.1 File Format and File Name.....	4-1
4.2 Grammar of C Source .....	4-2
4.2.1 Data Type .....	4-2
4.2.2 Library Functions and Header Files.....	4-3
4.2.3 In-line Assemble .....	4-4
4.2.4 Prototype Declarations .....	4-4
4.3 Grammar of Assembly Source .....	4-5
4.3.1 Statements.....	4-5
4.3.2 Notations of Operands.....	4-9
4.3.3 Extended Instructions .....	4-11
4.3.4 Preprocessor Directives.....	4-12
4.4 Precautions for Creation of Sources .....	4-13
<b>5 GNU17 IDE</b> .....	<b>5-1</b>
5.1 Overview .....	5-1
5.1.1 Features.....	5-1
5.1.2 Some Notes on Use of the IDE.....	5-1

5.2	Starting and Quitting the IDE .....	5-2
5.2.1	Starting the IDE .....	5-2
5.2.2	Quitting the IDE .....	5-3
5.3	IDE Window.....	5-4
5.3.1	Menu Bar .....	5-5
5.3.2	Window Toolbar.....	5-12
5.3.3	Editor Area.....	5-13
5.3.4	[C/C++ Projects] View .....	5-16
5.3.5	[Navigator] View .....	5-19
5.3.6	[Outline] View .....	5-22
5.3.7	[Console] View .....	5-24
5.3.8	[Problems] View .....	5-25
5.3.9	[Properties] View.....	5-26
5.3.10	[Make Targets] View .....	5-27
5.3.11	[Search] View .....	5-28
5.3.12	[Bookmarks] View .....	5-31
5.3.13	[Tasks] View .....	5-32
5.3.14	View Manipulation.....	5-33
5.3.15	Perspectives.....	5-36
5.4	Projects .....	5-37
5.4.1	What Is a Project?.....	5-37
5.4.2	Creating a New Project.....	5-37
5.4.3	Opening and Closing a Project .....	5-40
5.4.4	Switching Workspaces.....	5-41
5.4.5	Importing an Existing Project.....	5-42
5.4.6	Deleting a Project .....	5-45
5.4.7	Changing the Project Name.....	5-46
5.4.8	Resource Manipulation in a Project .....	5-47
5.4.9	File Filter .....	5-60
5.4.10	Working Set .....	5-61
5.4.11	Project Properties .....	5-65
5.5	The Editor and Editing Source Files.....	5-67
5.5.1	Starting the Editor .....	5-67
5.5.2	Basic Editing Facilities .....	5-69
5.5.3	Editing Functions for C Source Files.....	5-70
5.5.4	[Outline] View .....	5-77
5.5.5	Navigation History.....	5-78
5.5.6	Bookmarks.....	5-79
5.5.7	Tasks.....	5-84
5.5.8	Customizing the Editor.....	5-89
5.5.9	Using an External Editor.....	5-91
5.6	Search.....	5-93
5.6.1	Text Search .....	5-93
5.6.2	File Search.....	5-93
5.6.3	C Search.....	5-95
5.6.4	C Search from Context Menu .....	5-96
5.6.5	Canceling a Search .....	5-96
5.6.6	Search Results .....	5-97
5.7	Building a Program.....	5-99
5.7.1	Setting the CPU Type and Memory Model.....	5-99
5.7.2	Setting Compiler Options.....	5-101
5.7.3	Setting Assembler Options .....	5-108
5.7.4	Setting Linker Options.....	5-110
5.7.5	Generated Makefile.....	5-114
5.7.6	Editing a Linker Script.....	5-117
5.7.7	Executing a Build Process .....	5-141



5.7.8 Clean and Rebuild .....	5-142
5.7.9 Using an Original Makefile .....	5-143
5.8 Starting the Debugger .....	5-145
5.8.1 Generating a Parameter File .....	5-145
5.8.2 Setting the Debugger Startup Commands .....	5-149
5.8.3 Starting Up the Debugger .....	5-152
5.9 Customizing the IDE (Preferences) .....	5-154
5.10 Additional Description on Dialog Boxes .....	5-180
5.10.1 Properties for Project .....	5-180
5.10.2 Save Resources .....	5-211
5.10.3 Import > File system .....	5-212
5.10.4 Export > File system .....	5-214
5.10.5 Filters .....	5-216
5.10.6 Sorting .....	5-218
5.11 Files Generated in a Project by the IDE .....	5-219
<b>6 C Compiler .....</b>	<b>6-1</b>
6.1 Functions .....	6-1
6.2 Input/Output Files .....	6-1
6.2.1 Input File .....	6-1
6.2.2 Output Files .....	6-1
6.3 Starting Method .....	6-2
6.3.1 Startup Format .....	6-2
6.3.2 Command-line Options .....	6-2
6.4 Compiler Output .....	6-6
6.4.1 Output Contents .....	6-6
6.4.2 Data Representation .....	6-7
6.4.3 Method of Using Registers .....	6-9
6.4.4 Function Call .....	6-10
6.4.5 Stack Frame .....	6-10
6.4.6 Grammar of C Source .....	6-11
6.5 Functions of xgcc and Usage Precautions .....	6-12
<b>7 Library .....</b>	<b>7-1</b>
7.1 Library Overview .....	7-1
7.1.1 Library Files .....	7-1
7.1.2 Precautions to Be Taken When Adding a Library .....	7-2
7.2 Emulation Library .....	7-3
7.2.1 Overview .....	7-3
7.2.2 Floating-point Calculation Functions .....	7-4
7.2.3 Integral Calculation Functions .....	7-6
7.2.4 long long Type Calculation Functions .....	7-6
7.3 ANSI Library .....	7-7
7.3.1 Overview .....	7-7
7.3.2 ANSI Library Function List .....	7-7
7.3.3 Declaring and Initializing Global Variables .....	7-13
7.3.4 Lower-level Functions .....	7-14
<b>8 Assembler .....</b>	<b>8-1</b>
8.1 Functions .....	8-1
8.2 Input/Output Files .....	8-1
8.2.1 Input Files .....	8-1
8.2.2 Output File .....	8-2
8.3 Starting Method .....	8-3
8.3.1 Startup Format .....	8-3
8.3.2 Command-line Options .....	8-3
8.4 Scope .....	8-4

8.5	Assembler Directives.....	8-5
8.5.1	Text Section Defining Directive (.text) .....	8-5
8.5.2	Data Section Defining Directives (.rodata, .data).....	8-6
8.5.3	Bss Section Defining Directive (.bss).....	8-7
8.5.4	Data Defining Directives (.long, .short, .byte, .ascii, .space).....	8-8
8.5.5	Area Securing Directive (.zero).....	8-9
8.5.6	Alignment Directive (.align).....	8-10
8.5.7	Global Declaring Directive (.global).....	8-11
8.5.8	Symbol Defining Directive (.set).....	8-12
8.6	Extended Instructions.....	8-13
8.6.1	Arithmetic Operation Instructions.....	8-13
8.6.2	Comparison Instructions.....	8-15
8.6.3	Logic Operation Instructions.....	8-16
8.6.4	Data Transfer Instructions (between Stack and Register).....	8-17
8.6.5	Data Transfer Instructions (between Memory and Register).....	8-18
8.6.6	Immediate Data Load Instructions.....	8-19
8.6.7	Branch Instructions.....	8-21
8.6.8	Coprocessor Instructions.....	8-24
8.7	Optimization of Extended Instructions.....	8-25
8.8	Error/Warning Messages.....	8-29
8.9	Precautions.....	8-30
<b>9</b>	<b>Linker.....</b>	<b>9-1</b>
9.1	Functions.....	9-1
9.2	Input/Output Files.....	9-1
9.2.1	Input Files.....	9-1
9.2.2	Output Files.....	9-2
9.3	Starting Method.....	9-2
9.3.1	Startup Format.....	9-2
9.3.2	Command-line Options.....	9-2
9.4	Linkage.....	9-3
9.4.1	Default Linker Script.....	9-3
9.4.2	Examples of Linkage.....	9-4
9.5	Error Messages.....	9-6
9.6	Precautions.....	9-7
<b>10</b>	<b>Debugger.....</b>	<b>10-1</b>
10.1	Features.....	10-1
10.2	Input/Output Files.....	10-1
10.2.1	Input Files.....	10-1
10.2.2	Output Files.....	10-2
10.3	Starting the Debugger.....	10-3
10.3.1	Startup Format.....	10-3
10.3.2	Startup Options.....	10-3
10.3.3	Quitting the Debugger.....	10-4
10.4	Windows.....	10-5
10.4.1	Basic Window Configuration.....	10-5
10.4.2	[Source] Window.....	10-6
10.4.3	[Console] Window.....	10-18
10.4.4	[Registers] Window.....	10-19
10.4.5	[Memory] Window.....	10-22
10.4.6	[Breakpoints] Window.....	10-25
10.4.7	[Watch Expressions] Window.....	10-27
10.4.8	[Local Variables] Window.....	10-30
10.4.9	[Simulated I/O] Window.....	10-32
10.4.10	[Trace] Window.....	10-33

10.5	Method of Executing Commands .....	10-34
10.5.1	Entering Commands From the Keyboard.....	10-34
10.5.2	Parameter Input Format.....	10-35
10.5.3	Using Menus and Toolbar To Execute Commands .....	10-36
10.5.4	Using a Command File To Execute Commands .....	10-37
10.5.5	Log Files .....	10-38
10.6	Debugging Functions .....	10-39
10.6.1	Connect Modes.....	10-39
10.6.2	Loading a File .....	10-40
10.6.3	Source-Level Debugging Function.....	10-41
10.6.4	Manipulating Memory, Variables, and Registers .....	10-42
10.6.5	Executing the Program .....	10-45
10.6.6	Break Functions .....	10-49
10.6.7	Trace Functions.....	10-53
10.6.8	Simulated I/O .....	10-54
10.6.9	Flash Memory Operation .....	10-56
10.6.10	Support for Big Endian.....	10-59
10.7	Command Reference .....	10-60
10.7.1	List of Commands.....	10-60
10.7.2	Detailed Description of Commands .....	10-61
10.7.3	Memory Manipulation Commands .....	10-62
	c17 fb (fill area, in bytes).....	10-62
	c17 fh (fill area, in 16 bits).....	10-62
	c17 fw (fill area, in 32 bits).....	10-62
	x (memory dump).....	10-64
	set { } (data input) .....	10-66
	c17 mvb (copy area, in bytes).....	10-67
	c17 mvh (copy area, in 16 bits).....	10-67
	c17 mvw (copy area, in 32 bits) .....	10-67
	c17 df (save memory contents).....	10-69
	c17 readmd (memory read mode) .....	10-71
10.7.4	Register Manipulation Commands.....	10-72
	info reg (display register) .....	10-72
	set \$ (modify register) .....	10-73
10.7.5	Program Execution Commands .....	10-74
	continue (execute continuously).....	10-74
	until (execute continuously with temporary break) .....	10-76
	step (single-step, every line) .....	10-78
	stepi (single-step, every mnemonic) .....	10-78
	next (single-step with skip, every line) .....	10-80
	nexti (single-step with skip, every mnemonic).....	10-80
	finish (finish function) .....	10-82
	c17 callmd (set user function call mode) .....	10-83
	c17 call (call user function) .....	10-84
10.7.6	CPU Reset Commands .....	10-86
	c17 rst (reset).....	10-86
	c17 rstt (reset target) .....	10-87
10.7.7	Interrupt Commands .....	10-88
	c17 int (interrupt).....	10-88
	c17 intclear (clear interrupt) .....	10-89
	c17 int_load (load interrupt event file).....	10-90
10.7.8	Break Setup Commands.....	10-91
	break (set software PC break) .....	10-91
	tbreak (set temporary software PC break).....	10-91
	hbreak (set hardware PC break).....	10-94
	thbreak (set temporary hardware PC break) .....	10-94
	delete (clear break by break number) .....	10-97

clear (clear break by break position).....	10-99
enable (enable breakpoint).....	10-101
disable (disable breakpoint).....	10-101
ignore (disable breakpoint with ignore counts).....	10-103
info breakpoints (display breakpoint list).....	10-104
c17 timebrk (set lapse of time break).....	10-105
10.7.9 Symbol Information Display Commands.....	10-106
info locals (display local symbol).....	10-106
info var (display global symbol).....	10-106
print (alter symbol value).....	10-107
10.7.10 File Loading Commands.....	10-108
file (load debugging information).....	10-108
load (load program).....	10-109
c17 loadmd (set program load mode).....	10-110
10.7.11 Map Information Commands.....	10-111
c17 rpf (set map information).....	10-111
c17 map (display map information).....	10-112
10.7.12 Flash Memory Manipulation Commands.....	10-113
c17 fls (set flash memory).....	10-113
c17 fle (erase flash memory).....	10-114
10.7.13 Trace Command.....	10-115
c17 tm (set trace mode).....	10-115
10.7.14 Simulated I/O Commands.....	10-118
c17 stdin (data input simulation).....	10-118
c17 stdout (data output simulation).....	10-119
10.7.15 Flash Writer Commands.....	10-120
c17 fwe (erase program/data).....	10-120
c17 fwlp (load program).....	10-121
c17 fwdl (load data).....	10-122
c17 fwdc (copy target memory).....	10-123
c17 fwd (display flash writer information).....	10-124
10.7.16 Other Commands.....	10-125
c17 log (logging).....	10-125
source (execute command file).....	10-126
c17 clockmd (set execution counter mode).....	10-127
c17 clock (display execution counter).....	10-127
target (connect target).....	10-129
detach (disconnect target).....	10-130
pwd (display current directory).....	10-131
cd (change current directory).....	10-131
c17 firmupdate (update firmware).....	10-132
c17 ttbr (set TTBR).....	10-133
c17 help (help).....	10-134
quit (quit debugger).....	10-136
10.8 Parameter Files.....	10-137
10.9 Status and Error Messages.....	10-140
10.9.1 Status Messages.....	10-140
10.9.2 Error Messages.....	10-140
10.10 Embedded System Simulator (ES-Sim17).....	10-142
10.10.1 Input/Output Files.....	10-143
10.10.2 Starting and Terminating ES-Sim17.....	10-144
10.10.3 Simulating I/O Ports.....	10-145
10.10.4 Simulating SVD.....	10-147
10.10.5 Simulating an LCD Panel.....	10-148
10.10.6 ES-Sim17 Error Messages.....	10-150
10.10.7 Restrictions.....	10-150

<b>11 Other Tools</b> .....	<b>11-1</b>
11.1 make.exe.....	11-1
11.1.1 Functional Outline.....	11-1
11.1.2 Input File.....	11-1
11.1.3 Starting Method.....	11-2
11.1.4 make Files.....	11-3
11.1.5 Macro Definition and Reference.....	11-5
11.1.6 Dependency List.....	11-6
11.1.7 Suffix Definitions.....	11-9
11.1.8 clean.....	11-11
11.1.9 Messages.....	11-12
11.1.10 Precautions.....	11-12
11.2 ccap.exe.....	11-13
11.2.1 Function.....	11-13
11.2.2 Output File.....	11-13
11.2.3 Method for Using ccap.....	11-13
11.2.4 Error Messages.....	11-14
11.3 objdump.exe.....	11-15
11.3.1 Function.....	11-15
11.3.2 Input Files.....	11-15
11.3.3 Method for Using objdump.....	11-15
11.3.4 Dump Format.....	11-16
11.3.5 Error Message.....	11-19
11.3.6 Precautions.....	11-19
11.4 objcopy.exe.....	11-20
11.4.1 Function.....	11-20
11.4.2 Input/Output Files.....	11-20
11.4.3 Method for Using objcopy.....	11-21
11.4.4 Creating HEX Files.....	11-21
11.5 ar.exe.....	11-22
11.5.1 Function.....	11-22
11.5.2 Input/Output Files.....	11-22
11.5.3 Method for Using ar.....	11-23
11.6 moto2ff.exe.....	11-25
11.6.1 Function.....	11-25
11.6.2 Input/Output Files.....	11-25
11.6.3 Startup Format.....	11-25
11.6.4 Error/Warning Messages.....	11-26
11.6.5 Creating ROM Area Data.....	11-26
11.7 sconv32.exe.....	11-27
11.7.1 Function.....	11-27
11.7.2 Input/Output Files.....	11-27
11.7.3 Startup Format.....	11-27
11.7.4 Error Messages.....	11-28
11.8 Outline of the Development Tools.....	11-29
11.9 winfog.exe.....	11-30
11.9.1 Outline of winfog.....	11-30
11.9.2 Input/Output Files.....	11-30
11.9.3 Starting Up.....	11-31
11.9.4 Window.....	11-32
11.9.5 Menus and Toolbar Buttons.....	11-33
11.9.6 Operation Procedure.....	11-34
11.9.7 Error/Warning Messages.....	11-37
11.9.8 Sample Output File.....	11-38

## CONTENTS

11.10 winmdc.exe .....	11-39
11.10.1 Outline of winmdc .....	11-39
11.10.2 Input/Output Files .....	11-39
11.10.3 Starting Up.....	11-40
11.10.4 Menus and Toolbar Buttons .....	11-41
11.10.5 Operation Procedure .....	11-42
11.10.6 Error Messages .....	11-45
11.10.7 Sample Output File .....	11-46

# 1 General

## 1.1 Features

---

The S1C17 Family C Compiler Package contains software development tools for compiling C source files, assembling assembly source files, linking object files, debugging executable files, making mask data and other utilities. The tools are common to all the models of the S1C17 Family.

Its principal features are as follows:

### **Powerful optimizing function**

The C Compiler is designed to suit to the S1C17 architecture, it makes it possible to deliver minimized codes. The high-optimize ability does not lose most of the debugging information, and it enables C source level debugging.

### **Useful extended instructions are provided**

The extended instructions allow the programmer to describe assembly source simply without the need of knowing the data size. The immediate data extension using the "ext" instruction and some useful functions that need multiple basic instructions are described with an extended instruction.

### **C and assembly source level debugger with a simulator function**

The debugger supports C source level debugging and assembly source level debugging. By using an ICD, the program can be debugged even when the target board is operating. It also provides a simulator function that allows debugging on a personal computer without using hardware tools.

### **Integrated development environment for Windows**

Designed to run under Windows 2000 Professional and Windows XP, the **GNU17 IDE** is a seamless integrated development environment suitable for a wide range of development tasks, from source creation to debugging.

## 1.2 Tool Composition

---

### 1.2.1 Composition of Package

The S1C17 Family C Compiler Package contains the elements listed below. Please check to make sure that all elements are supplied.

- 1) Tool disks (CD-ROM) .....One
- 2) S5U1C17001C Manual (C Compiler Package for S1C17 Family) .....One each in English and Japanese
- 3) Warranty card .....One each in English and Japanese

### 1.2.2 Outline of Software Tools

The following shows the outlines of the principle tools included in the package.

#### (1) C Compiler (**xgcc.exe**)

This tool is made based on GNU C Compiler and is compatible with ANSI C. This tool invokes **cpp.exe** and **cc1.exe** sequentially to compile C source files to the assembly source files for the S1C17 Family. It has a powerful optimizing ability that can generate minimized assembly codes. The **xgcc.exe** can also invoke the **as.exe** assembler to generate object files.

#### (2) Assembler (**as.exe**)

This tool assembles assembly source files output by the C compiler and converts the mnemonics of the source files into object codes (machine language) of the S1C17 Core. The **as.exe** allows the user to invoke the assembler through **xgcc.exe**, this makes it possible to include preprocessor directives into assembly source files. The results are output in an object file that can be linked or added to a library.

#### (3) Linker (**ld.exe**)

The linker defines the memory locations of object codes created by the C compiler and assembler, and creates executable object codes. This tool puts together multiple objects and library files into one file.

#### (4) Debugger (**gdb.exe**)

This debugger serves to perform source-level debugging by controlling an ICD. It also comes with a simulator function that allows debugging on a personal computer.

The **gdb.exe** supports Windows GUI. Commands that are used frequently, such as break and step, are registered on the tool bar, minimizing the necessary keyboard operations. Moreover, various data can be displayed in multi windows, with a resultant increased efficiency in the debugging tasks.

#### (5) Librarian (**ar.exe**)

This tool is used to edit libraries. The **ar.exe** can register object modules created by the C compiler and assembler to libraries, delete object modules in libraries and restore library modules to the original object files.

#### (6) Make (**make.exe**)

This tool automatically executes from compile to link according to the command lines described in the make file. The basic make file can be created by the **IDE**.

#### (7) GNU17 IDE (**eclipse.exe**)

The development workbench provides an integrated development environment for a wide range of development tasks, from source creation to debugging.

This package contains other gnu tools, sample programs and several utility programs. For details on those programs, please refer to "readmeVxx.txt" (xx indicates version) on the disk.

**Note:** Only the command options for each tool described in the respective section are guaranteed to work. If other options are required, they should only be used at the user's own risk.



# 2 Installation

This chapter describes the required working environments for the tools supplied in the S1C17 Family C Compiler Package and their installation methods.

## 2.1 Working Environment

---

To use the S1C17 Family C Compiler Package, the following conditions are necessary:

### Personal computer

An IBM PC/AT or a compatible machine which is equipped with a CPU equal to or better than a Pentium4 1.50 GHz, and 512MB or more of memory is recommended.

To use an optional ICD, a USB port is required.

### Display

A display unit capable of displaying 1,024 × 768 dots or more is recommended.

**Note:** Selecting an ultra-large font and high contrast in the Windows "Display Properties" may prevent proper display of the **IDE** screen. Furthermore, "Color quality" should be set to 16 bits or higher.

### Hard drive

The hard drive must have at least 500MB of empty space to install the S1C17 Family C Compiler Package.

### CD-ROM drive

Since the installation is done from a CD-ROM, a CD-ROM drive is required.

### Mouse

A mouse is necessary to operate the tools.

### Debugging tool

To debug the program and the target system, an optional ICD is needed in addition to this software package.

### System software

The tools support Microsoft Windows 2000 Professional or Windows XP (English or Japanese version).

### Other

- Please go through the precautions and restrictions given in "readmeVxx.txt" (English, Japanese) (xx indicates version) on the disk.
- Running the tools in this package presumes the presence of **cygwin1.dll**. Although **cygwin1.dll** is stored in the \gnu17 directory, if **cygwin1.dll** is already installed on your system, the duplication may cause problems. If so, remove the copy of **cygwin1.dll** installed in your system or exclude it from the environment variable PATH settings to ensure that the file referenced is always the copy of **cygwin1.dll** located in the \gnu17 directory.

## 2.2 Installation Method

### Installing the tools

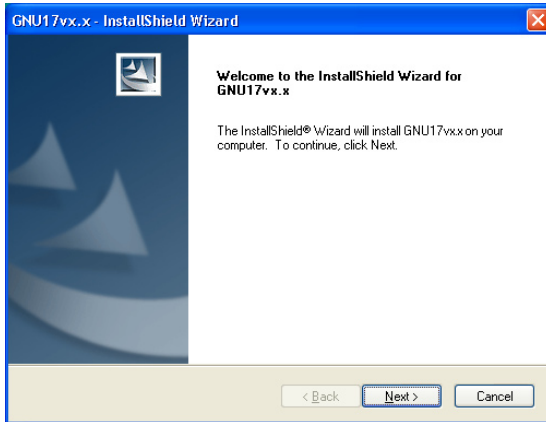
To install the tools, run the installer found on the CD-ROM provided.

(1) Start Windows 2000/XP.

If Windows is already running, close all other programs that are currently open.

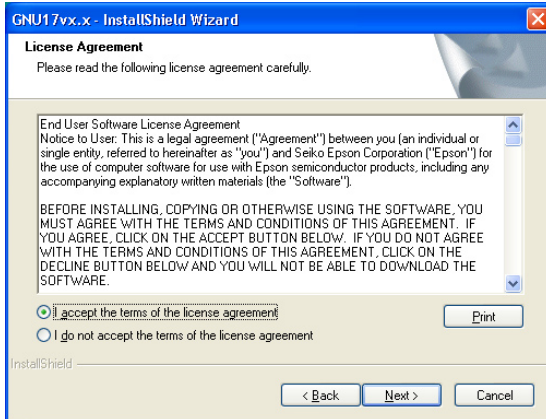
(2) Insert the CD-ROM into the drive and open the root directory to display its contents.

(3) Double-click **Setup.exe** to launch the installer.



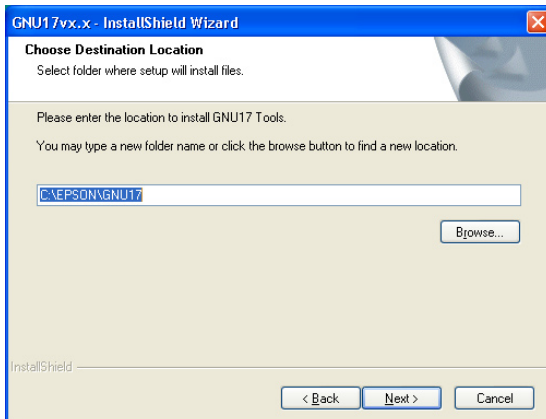
You will see the install wizard start screen.

(4) Click the [Next >] button to go to the next step.



Read the end user software license agreement displayed on the following screen.

(5) If you agree to the terms of the license, select "I accept the terms of the license agreement" and click the [Next >] button. If you do not agree, click the [Cancel] button to close the installer.

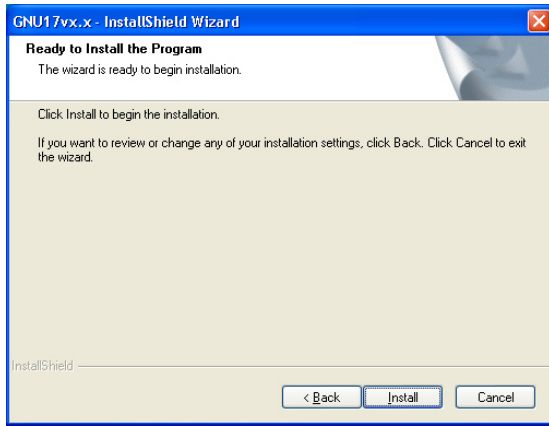


The screen displayed allows you to select the directory into which the gnu17 tools are to be installed.

(6) Check the destination directory in which the tool will be installed.

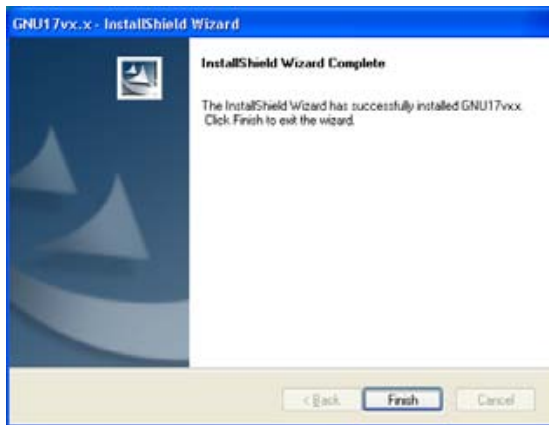
To switch to a different directory, use the [Browse...] button to bring up a directory selection dialog box. From the list in this dialog box, select the directory in which you want to install the tools, or enter a path to the desired directory in the [Path] text box. Click the [OK] button.

(7) Click the [Next >] button.



This is the install start screen.

(8) Click the [Install] button to begin installing.

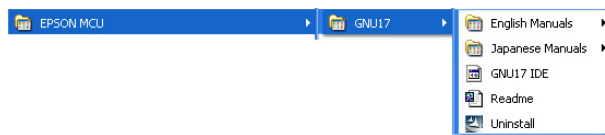


When installation is completed, a complete screen is displayed.

(9) Click the [Finish] button to quit the installer.

This completes installation of the tools.

Assuming installation finished successfully, an [EPSON MCU] > [GNU17vx.x] menu will be added to the Windows startup menu.



## 2 INSTALLATION

### Installed files

The following lists the configuration of directories and files after copying.

\\EPSON (default)

  \\gnu17 (Root DIR of the gnu17 tool)

    readmeVxx.txt

    Copying.GNU

    xgcc.exe, cpp.exe, cc1.exe

    xgcc\_filt.exe

    as.exe

    ld.exe

    ar.exe

    gdb.exe

    make.exe

    objdump.exe

    objcopy.exe

    moto2ff.exe

    sconv32.exe

    rm.exe

    sed.exe

    cp.exe

    sh.exe

    ccap.exe Console capture utility

    diff.exe

    gdbtk.ini

    reset.gdb

    userdefine.gdb

    cygitc130.dll, cygitk30.dll, cygtc180.dll,

    cygtk80.dll, tix4180.dll

    cygwin1.dll

  \\essim17

  \\eclipse

    eclipse.exe

    eclipse.ini

    .eclipseproduct

    epl-v10.html

    notice.html

    startup.jar

    gnu17\_32\_trans.ico

    \\configuration

    \\cpuinfo

    \\features

    \\jre

    \\plugins

    \\readme

  \\lib

    \\16bit

      libc.a

      libgcc.a

      libstdio.a

    \\24bit

  \\include

  \\sample

  \\dev

  \\tool

  \\utility

  \\doc Manual and other documents

Information about tools (in English and Japanese)  
with xx indicating version.

GNU copyright

C compiler

C source shift JIS filter

Assembler

Linker

Librarian

Debugger

make

Object file information display utility

Object file copy and translate utility

ROM area file generation (ff filling) tool

Motorola S format converter

File remove utility

Stream editor

File copy utility

Bourn shell utility

Difference display utility

Debugger setup file

Debugger command file for the [Reset] button

Debugger command file for the [User Command] button

dll files for debugger

dll file for development tools

Embedded system simulator

GNU17 IDE execution file

Eclipse setup file

Eclipse version

EPL license

Software agreement

Eclipse startup library

gnu17 icon file

Startup configuration file, etc.

S1C17 configuration files

Features

Java virtual machine

Plugins

Release notes

Libraries

Libraries for 16-bit address space

ANSI C library file

Emulation library file

Simulated I/O library file

Libraries for 24-bit address space

(configured in the same way as \\16bit)

C include files

Sample files

Development tools

Middleware and other tools

Utilities

Refer to the "readmeVxx.txt" for the contents of the "sample" and "utility" directories.

### Precaution when installing over existing version

If the gnu17 tools have been installed over the old version, changes in the **GNU17 IDE** may not be reflected in the new version just installed. If this happens, temporarily close **GNU17 IDE**, then start from the command line prompt, as described below.

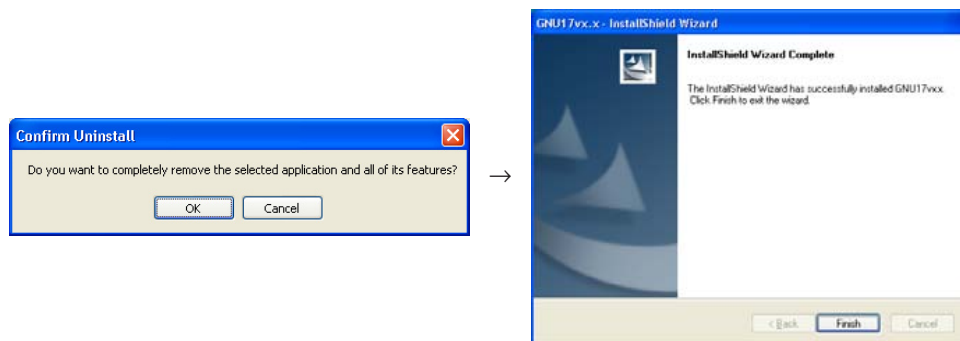
```
C:\EPSON\gnu17\eclipse>eclipse.exe -clean
```

### Precautions on setting the OS

- Select to "regular" font size in "Display Properties".
- When using a drive on the network as the tool and/or work drive, be sure to assign a drive name to it. The network name cannot be used.
- Do not use the USB port for the ICD in other drivers and applications. Furthermore, make sure that the port has been enabled when using a note PC as some can disable USB port.
- If the **gdb** debugger or **GNU17 IDE** have a problem on the GUI that causes an abnormal display, decrease the function level of the graphics or use a low-level standard display driver which has been supplied in the Windows package.

### Uninstalling the tools

To uninstall the tools, select [UnInstall] from [EPSON MCU] > [GNU17] in Windows startup menu, then click the [OK] button in the subsequent dialog box.



You also can use Add/Remove Programs in the Control Panel to uninstall the tools.

**Note:** If you set the \EPSON\gnu17\eclipse\workspace directory in the workspace, make a backup of the workspace directory before removing the tools. (Projects are saved to this directory.)

### About the license

#### GNU

The C compiler tools in this package is made based on the GNU C Compiler designed by Free Software Foundation, Inc. Please read the "Copying.GNU" text file for the license before using.

#### EPL

**GNU17 IDE** complies with the Open Source Initiative EPL (Eclipse Public License) 1.0. For more information on the EPL, refer to epl-v10.html in the \gnu17\eclipse directory.

THIS PAGE IS BLANK.

# 3 Software Development Procedures

## 3.1 Software Development Flow

Figure 3.1.1 shows typical software development flow.

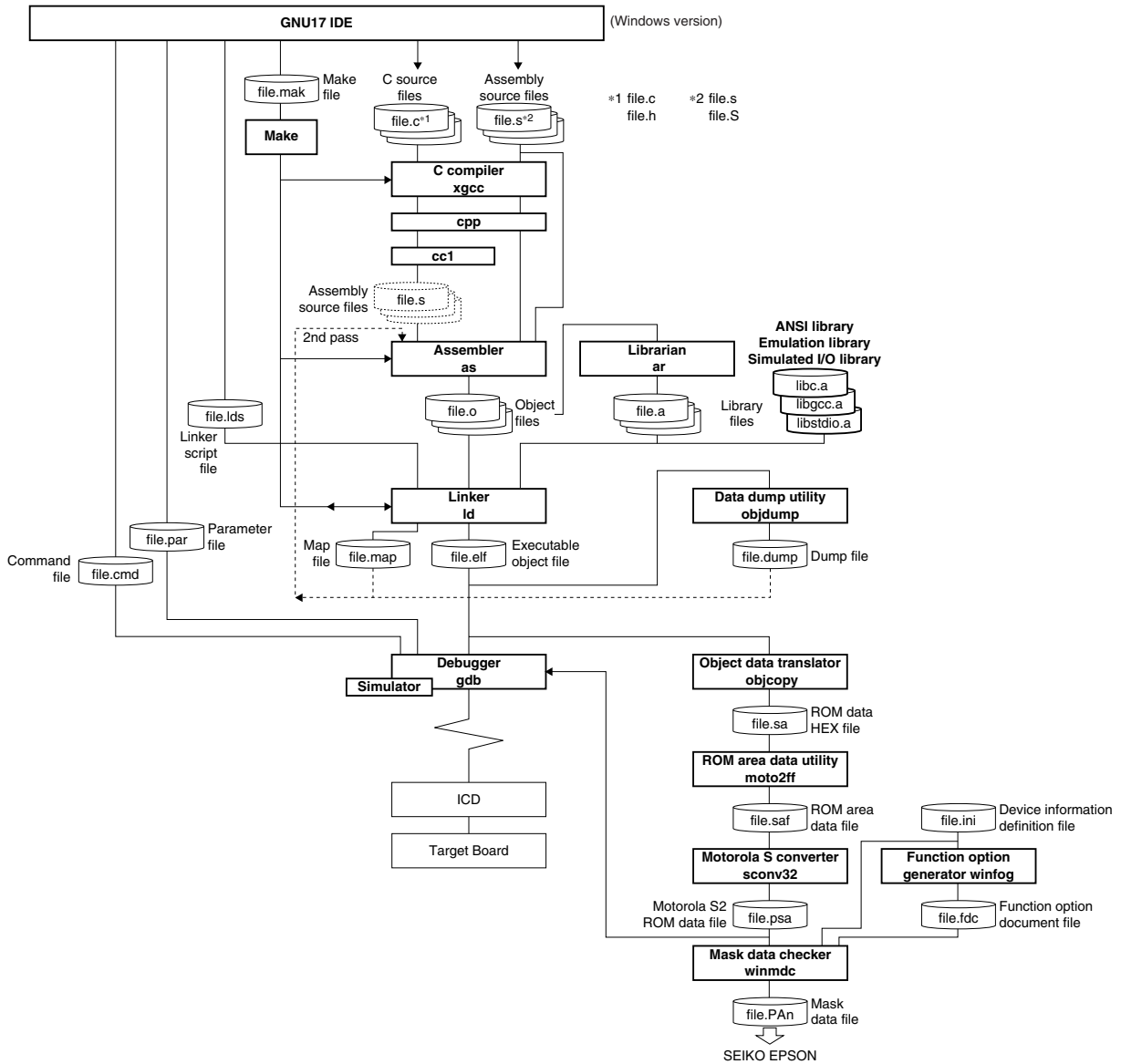


Figure 3.1.1 Software development flow

### 3 SOFTWARE DEVELOPMENT PROCEDURES

As shown above, the tools included with this package handle all software processing required after creating a source program. All basic operations except debugging are performed in the **GNU17 IDE** (hereafter the **IDE**). The development flow is outlined below.

#### (1) Creating a project

Use the **IDE** to create a new project. The system will set up the project file needed to collectively manage the software resources of the application to be developed and a workspace directory in which those resources are stored.

#### (2) Creating a source program

Use the **IDE** editor or a general-purpose editor to create a source file and add it to the project.

#### (3) Building a program

Start by using the **IDE** to set startup options for the tools from the C compiler to the linker and linker scripts. Then execute a build process from the **IDE**. The system will execute **make.exe** using the makefile (generated according to the set content), generating object files in debuggable 'elf' format. The necessary processing is automatically executed sequentially in the following operations according to the makefile.

- Compile (for C sources)

The source files are compiled by the **xgcc** C compiler, generating the object files (.o) are to be input to the **ld** linker.

- Assemble (assembler sources)

The assembler source files are assembled by the **as** assembler to generate the object files (.o) to be input to the **ld** linker.

If the source files include preprocessor instructions, use **xgcc** to perform preprocessing and assembly. When the necessary options are specified, **xgcc** will execute the **cpp** preprocessor and the **as** assembler.

- Link

The compilation and assembly operations described above will prepare one or multiple object files required for subsequent processing. The **ld** linker then generates an executable object file capable of being loaded and executed in the target ROM, namely 'elf' format object files that include information required for debugging, etc.

#### (4) Debugging

Use the 'elf' format object files generated by the **ld** linker to perform verification and debugging with the **gdb** debugger. Although an ICD can be used to debug hardware as well as software operation, the **gdb** has simulator mode that allows the PC to emulate device operations as the S1C17 Core and memory models. Debugger setting and startup can be performed from the **IDE**.

#### (5) Creating ROM data and mask data

Use the object file format conversion utility **objcopy** to create HEX files for writing the program into external and internal ROMs from the 'elf' format object files generated by the **ld** linker. Then, convert the HEX file for the internal ROM into a Motorola S2 file in which the unused area is filled with 0xff using **moto2ff** and **sconv32**. After creating the ROM data file according to the above procedure, be sure to perform the final verification of program operation on the actual target board using that file. Finally, pack the verified ROM data file and the mask option data file generated by **winfo** into a mask data file using **winmdc** and present it to Seiko Epson.

In addition to the tools described above, the C compiler package comes with the **ar** librarian. This tool organizes modules for general-purpose processing (e.g., object files output by the **as** assembler) as a library, facilitating future applications development involving the S1C17 Family.



## 3.2 Software Development Using the IDE

This section describes software development procedures using the **IDE** separately in several different cases. The actual operations are detailed in other tutorial sections in this manual.

First, before starting software development with the **IDE**, create a folder labeled "project" for each application. Use this folder to manage necessary resources.

If no projects are created in the **IDE**, software development with the **IDE** will start with project creation. The same applies when creating an entirely new application or when using one of programs created with an earlier version of the S1C17 tools.

If a project has already been created in the **IDE**, it is possible to migrate projects from another environment or to upgrade program versions by importing that project folder.

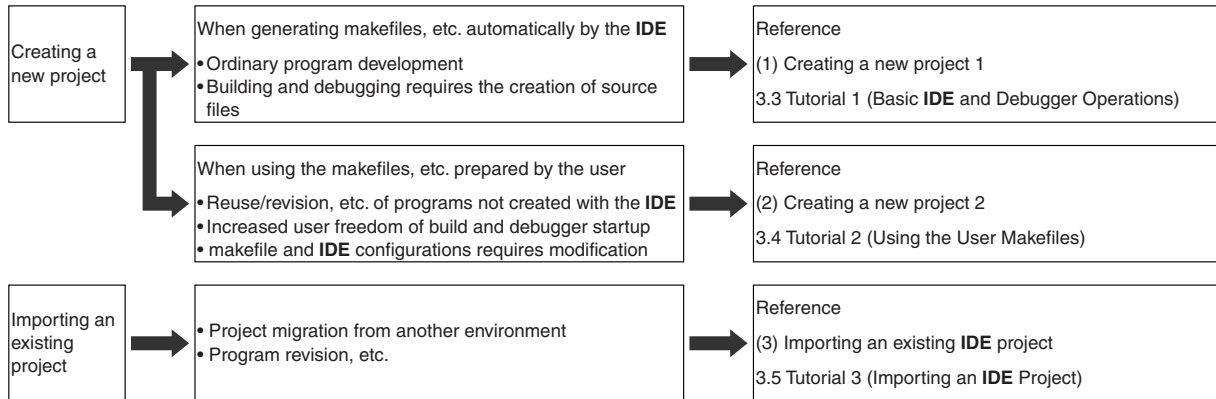


Figure 3.2.1 Software development with the **IDE**

(1) Creating a new project 1

This is the conventional procedure for developing software with the **IDE**. The user creates source files, after which the **IDE** automatically generates all other files required for build processing and debugger startup. The basic procedural flow is given below.

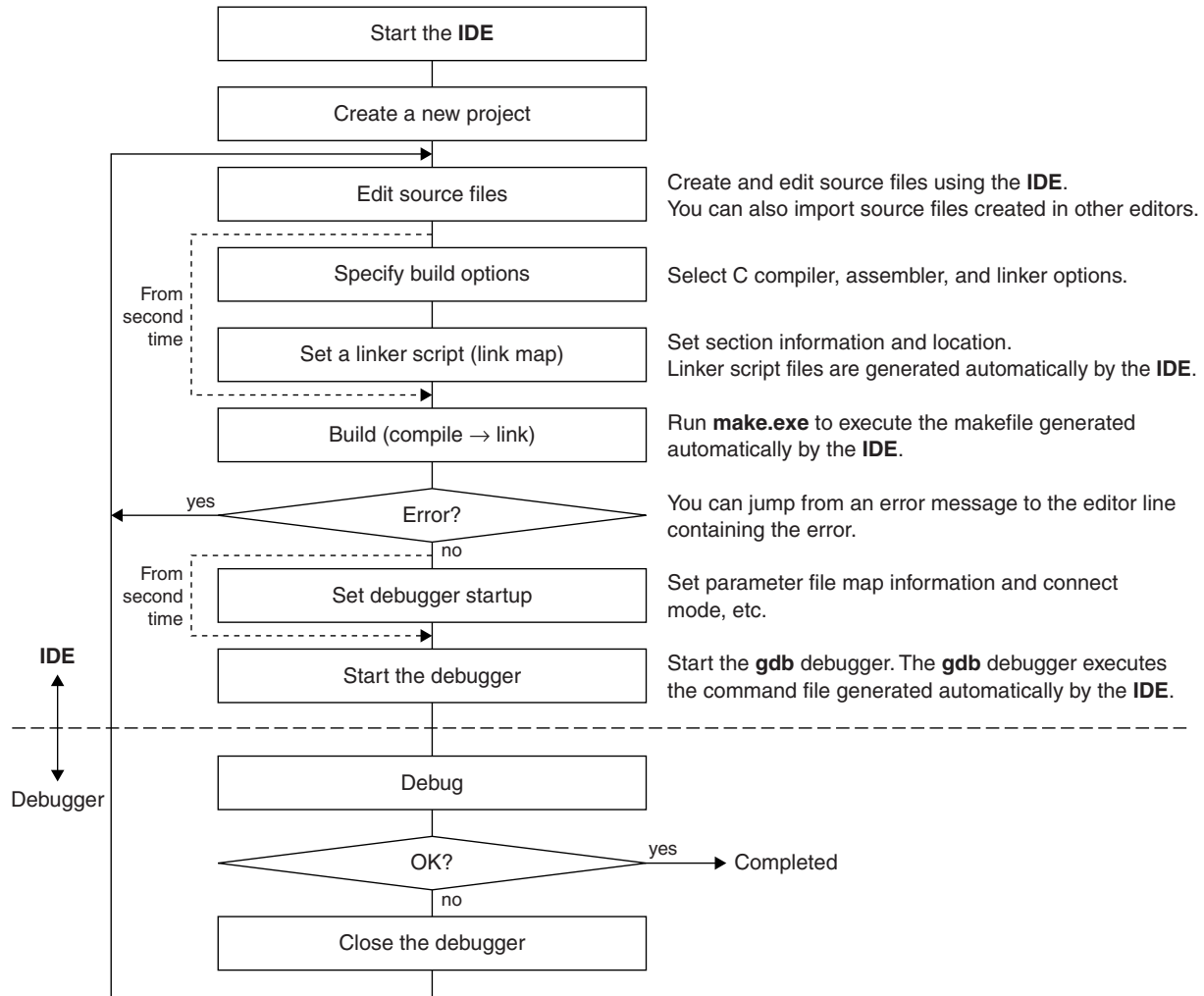


Figure 3.2.2 Procedural flow (makefiles, etc. generated automatically by the **IDE**)

For detailed information on basic operations, from starting the **IDE** to debugging the program, refer to Section 3.3, "Tutorial 1 (Basic **IDE** and Debugger Operations)".

**(2) Creating a new project 2**

When developing new software, a user makefile or a debug command file may be used instead of generating files automatically with the **IDE**.

The basic procedural flow is given below.

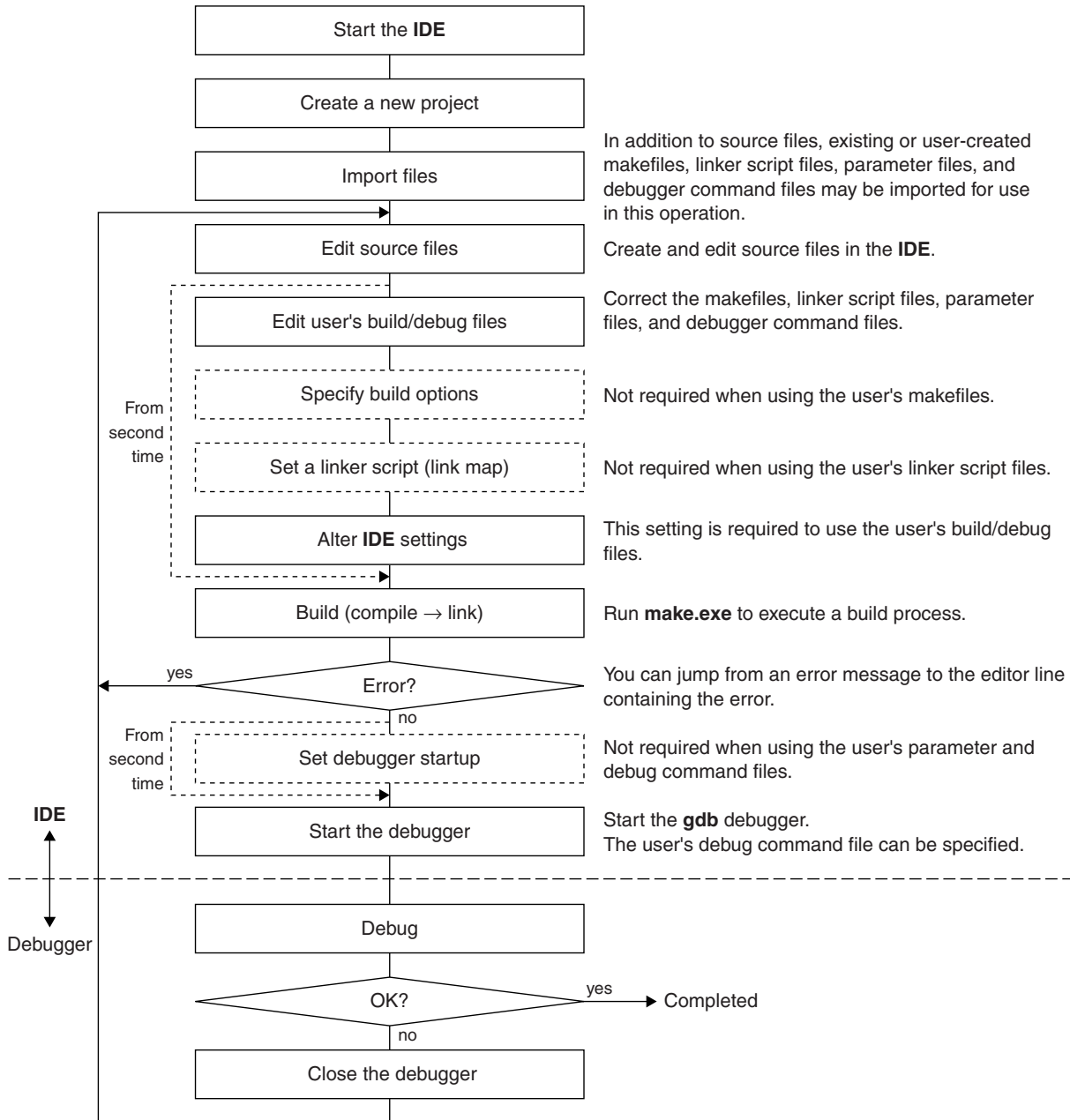


Figure 3.2.3 Procedural flow (using the user's makefiles, etc.)

For detailed information on building a program with the **IDE**, refer to Section 3.4, "Tutorial 2 (Using the User Makefiles)", which describes the procedure for building a sample program using a user makefile.

To use user makefiles, you must correct the makefile itself and alter the settings made in the **IDE**. Unless doing so would result in problems, we recommend using the files automatically generated by the **IDE**.

### (3) Importing an existing IDE project

If you have an existing project, you can simply import the project to continue working on your development or revisions. The project properties are inherited, so that re-configuration or other such operations are not required unless you intend to change them. However, project management files must remain intact in the project folder to be able to import projects.

The basic procedural flow is given below.

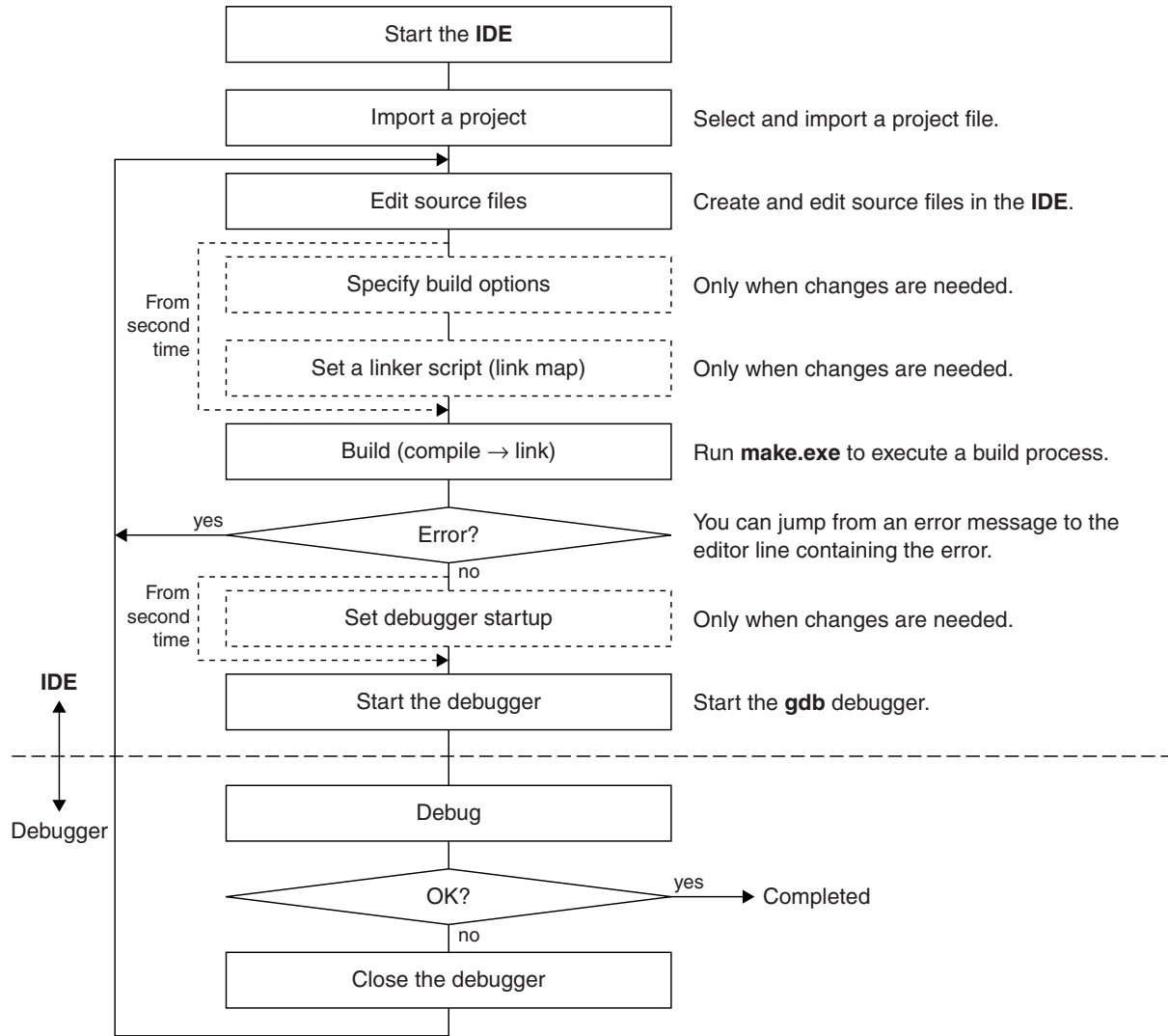


Figure 3.2.4 Procedural flow (importing an IDE project)

For detailed information on how to import a project, refer to Section 3.5, "Tutorial 3 (Importing an IDE Project)", which describes the procedure for importing a sample program created with the IDE.

## 3.3 Tutorial 1 (Basic IDE and Debugger Operations)

This section provides a tutorial on developing software with the **IDE**. For detailed information on each tool, refer to the sections in which the respective tools are described.

### Files used

This discussion assumes that the sample source files listed below are present in the `c:\EPSON\gnu17\sample\S1C17common\simulator\tst` directory.

<code>boot.s</code>	Assembler source file
<code>main.c</code>	C source file

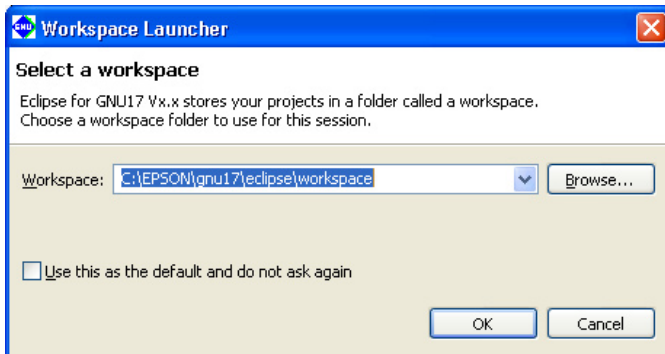
Described below is range of operations for creating a project, building a program, and verifying program operation using the above two source files. Note that this discussion assumes that you are using the **IDE** for the first time after installing the tools. If you have taken any actions in the **IDE**, the example screens may not match the ones you see on your PC.

### 3.3.1 Starting the IDE



**Step 1:** Double-click the **eclipse.exe** icon in the `c:\EPSON\gnu17\eclipse` directory to start the **IDE**. You also can start the **IDE** by selecting [EPSON MCU] > [GNU17] > [GNU17 IDE] from the Windows Start menu.

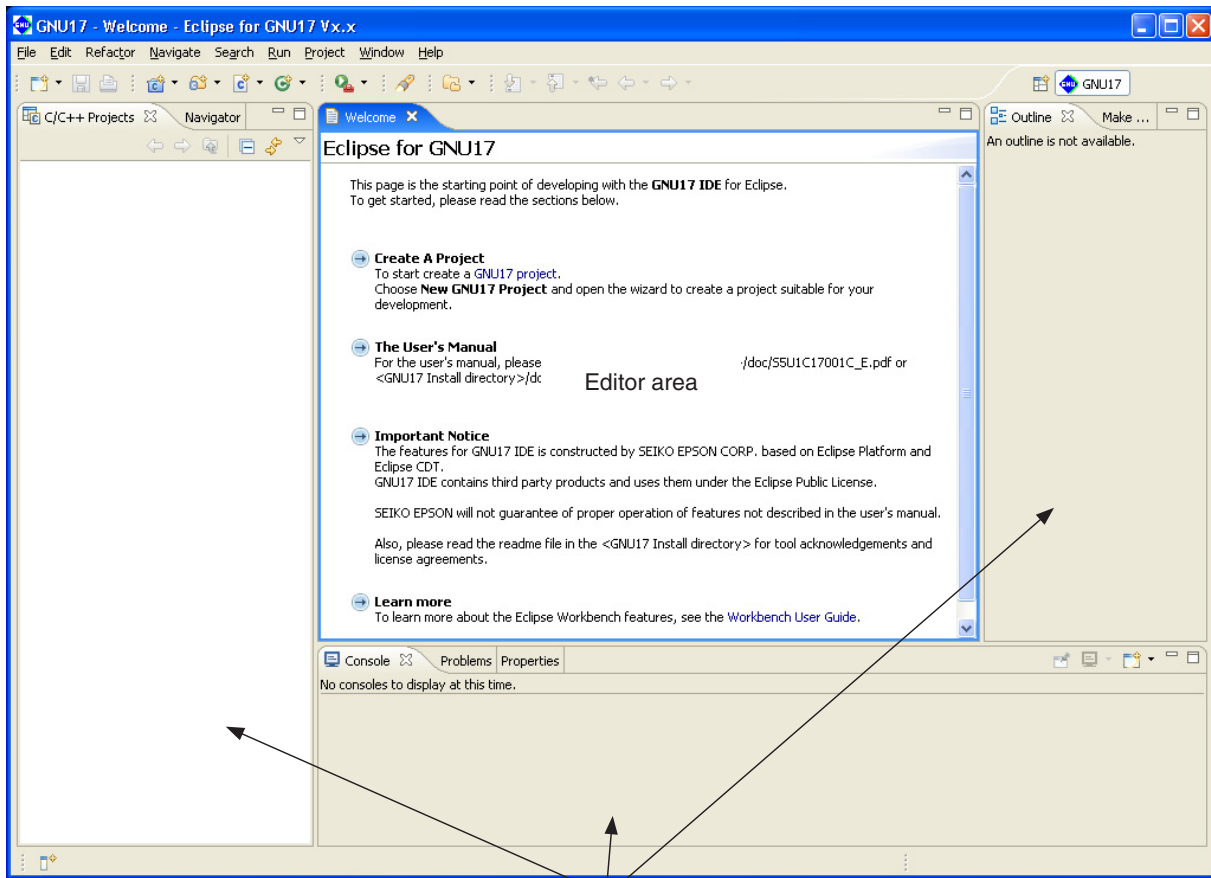
After an Eclipse splash screen, the [Workspace Launcher] dialog box shown below will appear. Specify the workspace (directory) in which you want to save the project resources and output files.




This tutorial uses the default workspace directory. You can select any directory or create a new directory and set it as the workspace.

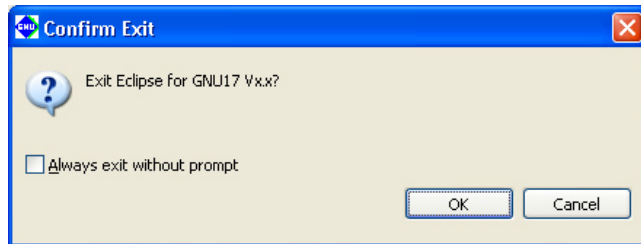
**Step 2:** Click the [OK] button.

The **IDE** window shown below will be displayed.



View

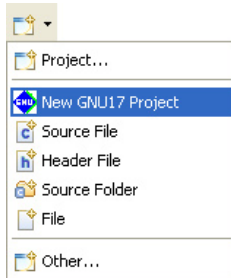
To quit before completing the tutorial, select [Exit] from the [File] menu of the **IDE**. Or use the window's  (close) button. When the following dialog box appears, click the [OK] button to quit or the [Cancel] button to cancel quitting.



### 3.3.2 Creating a Project

In applications development, a single executable program file is created from multiple source files. To manage these files in one location, you must create a project. The **IDE** generates programs on a per-project basis. In a sense, the project is the application program you want to develop, but the project actually created is a directory with a specified project name, wherein files containing project information (`.cdtproject`, `.gnu17project`, and `.project`) are generated.

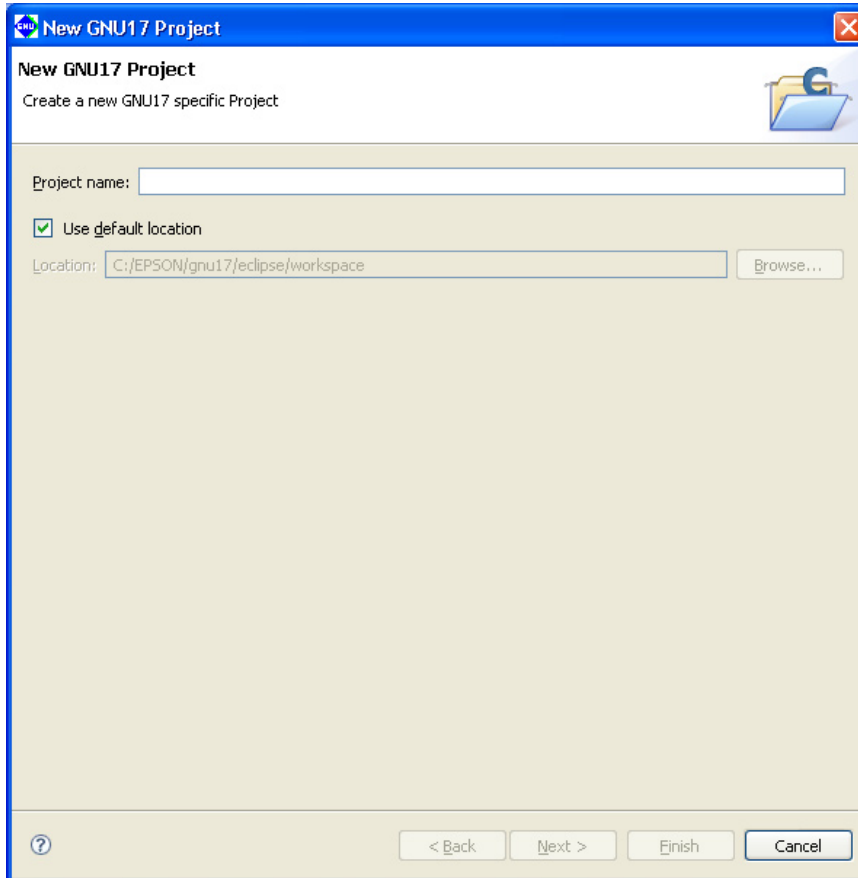
#### To create a new project



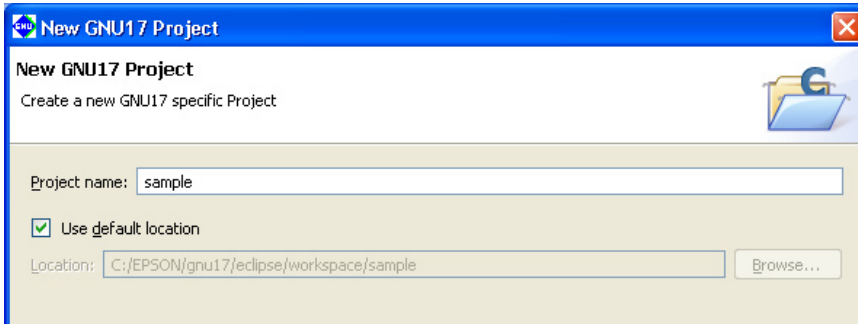
Step 3: Select [New GNU17 Project] from the [New] pulldown menu in the toolbar.

You can also select [New GNU17 Project] from the [File] menu or from [New] on the context menu (displayed by right-clicking) in the [C/C++ Projects/Navigator] view.

The [New GNU17 Project] wizard will start.



## Specifying a project name



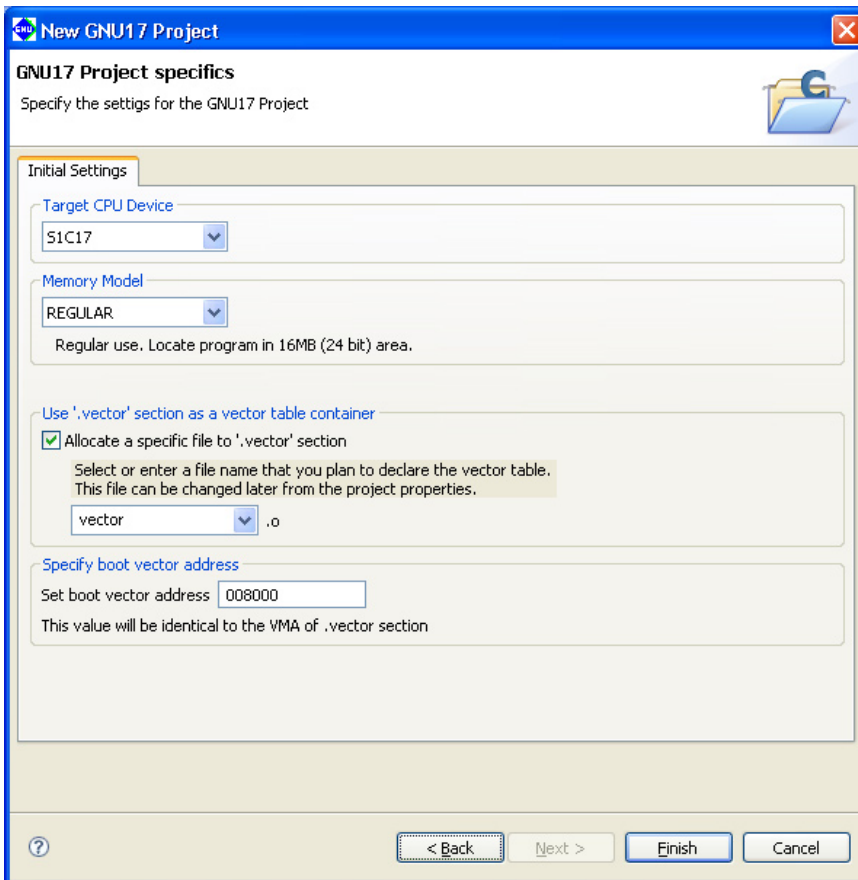
Step 4: Enter the project name "sample" in the [Project name:] text box.

Leave the [Use default location] check box selected. A project folder named "sample" will be generated in the workspace directory you specified when the **IDE** started.

The executable object file (.elf) generated when building a project is assigned the name you specify here.

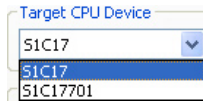
Step 5: Click the [Next>] button.

The system will go to the next screen, where you select a target CPU, memory model and vector table file.



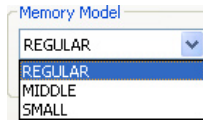


### Specifying a target CPU



Step 6: From the [Target CPU Device] combo box, select the target processor. Here, select "S1C17".

### Selecting a memory model



Step 7: Select the memory model supported by the processor from the [Memory Model] combo box. Here, select "REGULAR".

REGULAR: 24-bit address space (16MB)

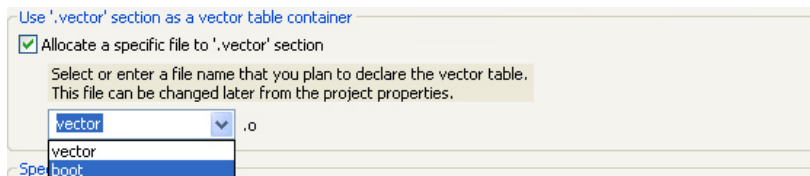
MIDDLE: 20-bit address space (1MB)

SMALL: 16-bit address space (64KB)

### Specifying a vector table file

The IDE requires the definition of a vector table section labeled `.vector` in a linker script to ensure that the trap vectors located in memory always begin with the trap table base address. In the [Use `.vector` section as a vector table container] field of this screen, specify whether to locate a specific object in the `.vector` section by selecting or unselecting the check box and set an object file name in the combo box. Here, we'll proceed assuming that `boot.o` is located in the `.vector` section.

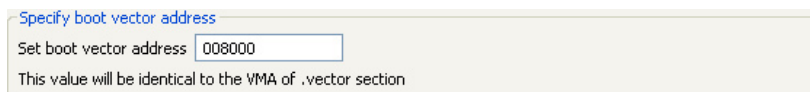
Step 8: Select `boot.o` from the pulldown list.



For detailed information on the `.vector` section, refer to Section 5.7.6, "Editing a Linker Script".

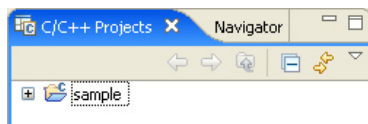
### Specifying a boot vector address

In the [Specify boot vector address] field, specify a boot vector address. The default boot vector address is "008000". The value set here will be used as the parameter for the TTBR setting command that will be written in the debugger startup command file created by the IDE as well as it will be used as the VMA of the `.vector` section that will be written in the linker script file. It is not necessary to alter the default value.



Step 9: Click the [Finish] button.

The [New GNU17 Project] wizard will be closed, creating a project with the specified name.



The target CPU, memory model and vector table file can be revised later.

### 3.3.3 Creating, Adding, and Editing a Source File

The IDE supports C and assembler to allow generation of an object from source files created in those languages. All source files required to generate an object must be added to the project created earlier.

#### Creating a source file

Use the IDE editor or a general-purpose editor to create a source file. You can also use an existing source file in the application you created for the S1C17 Family.

In this tutorial, we will use the source files prepared as examples.

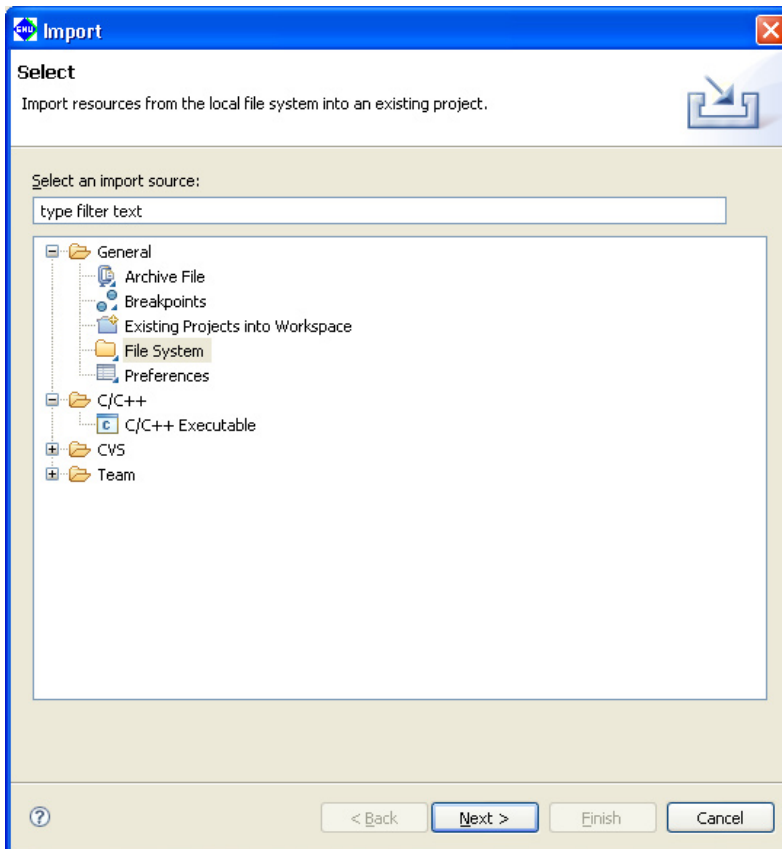
For detailed information on creating a new source file with the IDE, refer to Section 5.5, "The Editor and Editing Source Files".

#### Adding a source file

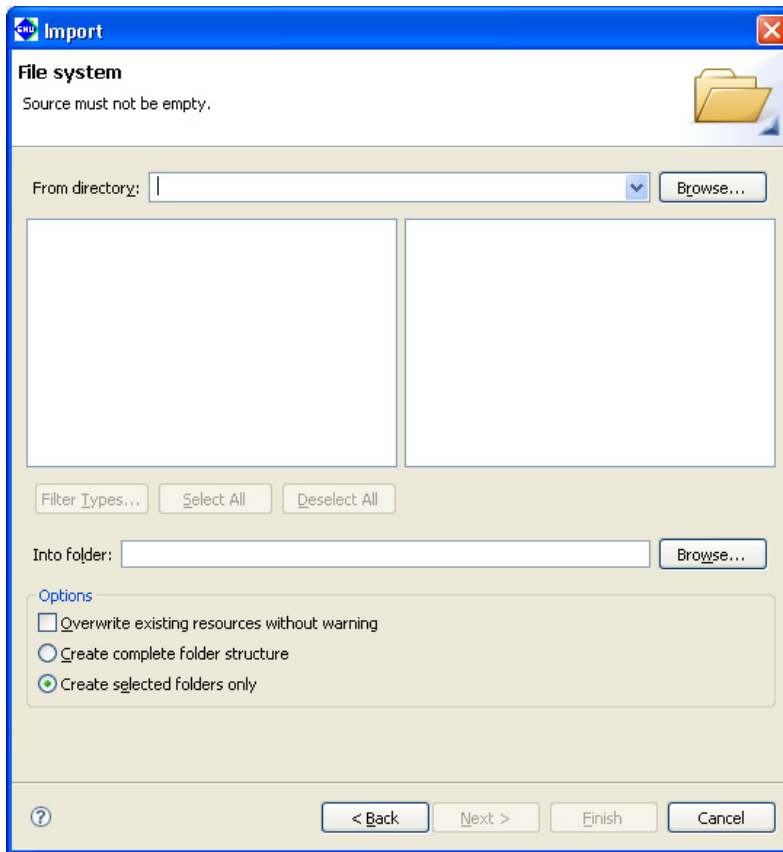
Load the source files prepared as samples into the project.

Step 10: Select [Import...] from the [File] menu.

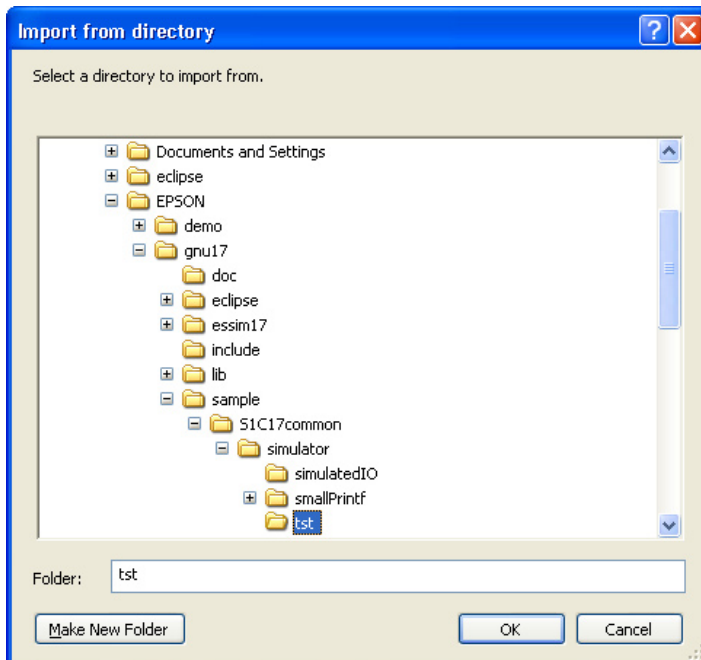
The [Import] wizard will start.



Step 11: From the list displayed, select [General] > [File System] and click the [Next>] button.

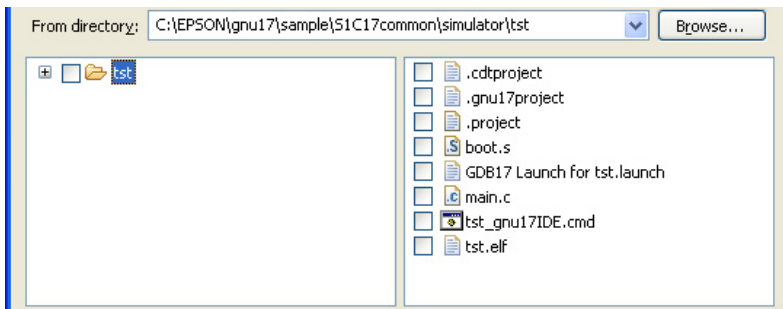


Step 12: Click the [Browse...] button for [From directory:]. The [Import from directory] dialog box will be displayed, so select the \EPSON\gnu17\sample\S1C17common\simulator\tst directory from the drive (C) in which you installed the **IDE** and click [OK].



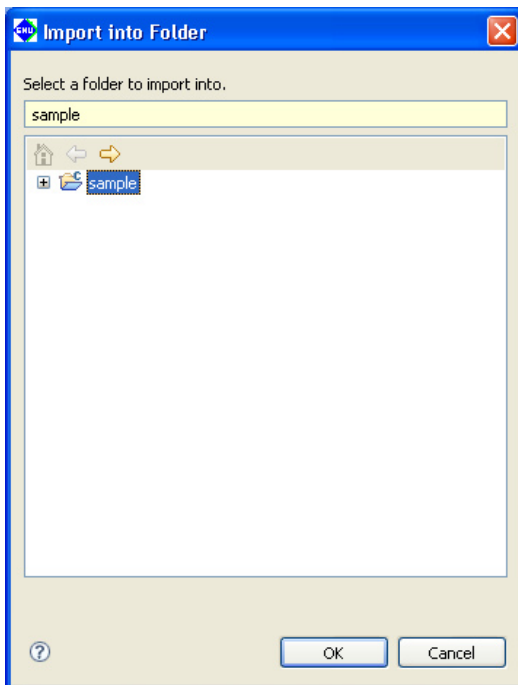
### 3 SOFTWARE DEVELOPMENT PROCEDURES

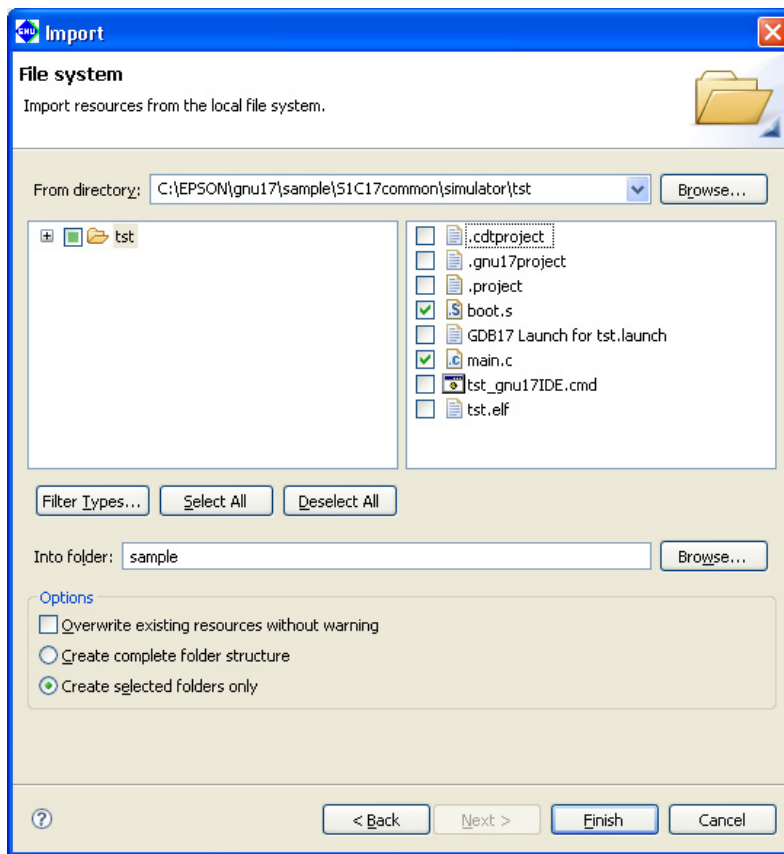
The directory you selected is displayed in the left-side list box, while the files contained in the directory are listed in the right-side list box.



Step 13: Select "boot.s" and "main.c" from the file list. Click to select the check box shown before the file name (flagged by a check mark when selected).

Step 14: Click the [Browse...] button for the [Into folder:]. This displays the [Import into Folder] dialog box. Select the "sample" folder and click [OK].

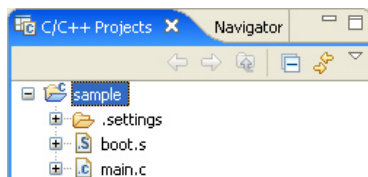




Step 15: After confirming that the dialog box is filled out as shown above, click the [Finish] button.

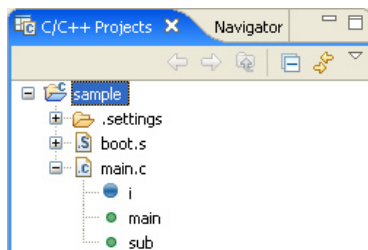
This procedure adds "boot.s" and "main.c" to the project.

Step 16: Double-click "sample" in the [C/C++ Projects] view, or click [+] shown before "sample".



The added source files are displayed in the "sample" folder in the [C/C++ Projects] view.

Step 17: Click [+] for "main.c" in the [C/C++ Projects] view.

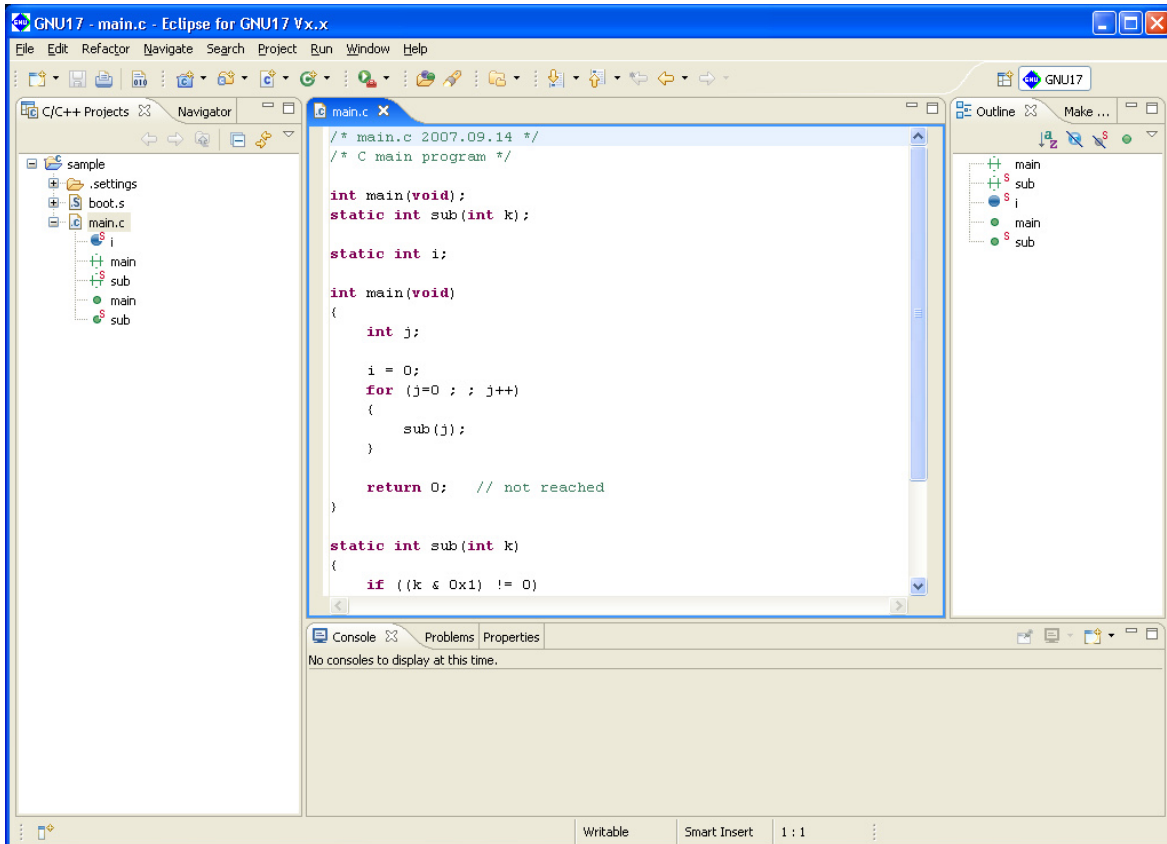


The global variables and functions defined in the file are displayed for C sources.

## Displaying and editing source files

Use the IDE editor to display and edit source files added to the project.

Step 18: Double-click "main.c" in the [C/C++ Projects] view.

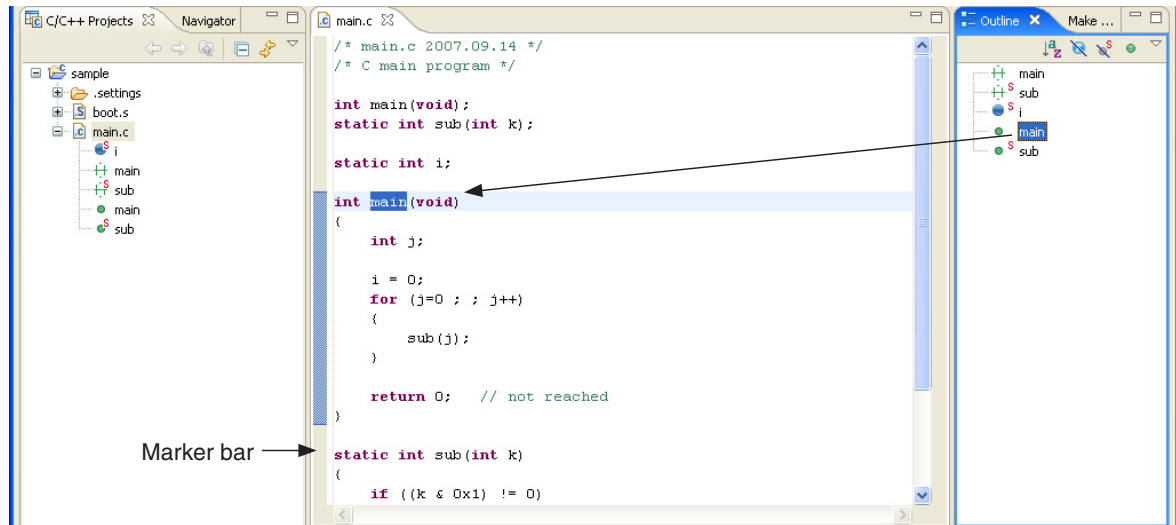


The contents of main.c are displayed in the editor. Here, you can correct the source as with a general-purpose editor. Furthermore, you can set up the editor so that selected files will be opened in a general-purpose editor you normally use.

For detailed information, refer to Section 5.5, "The Editor and Editing Source Files".

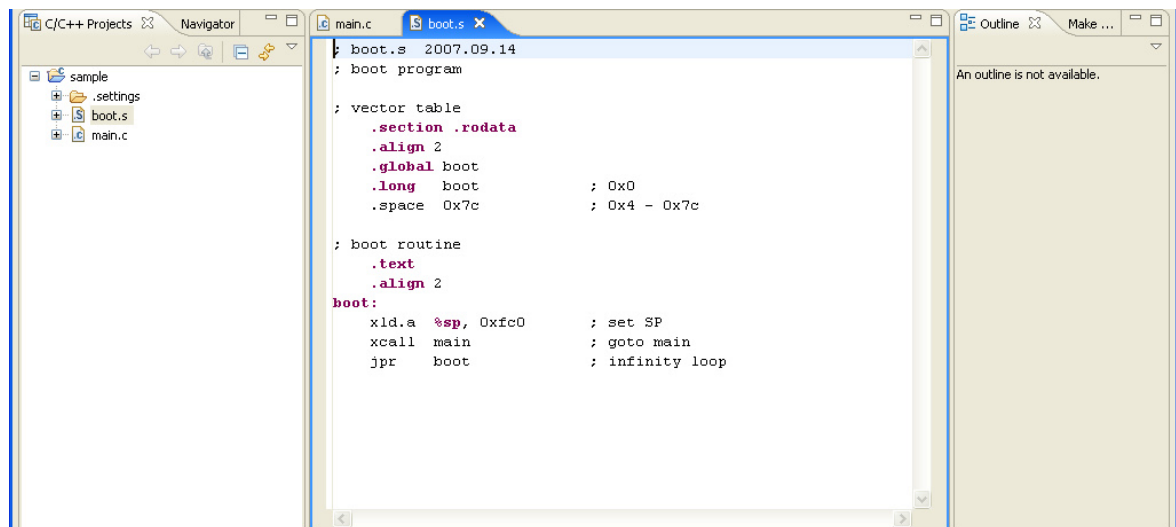
If C sources are displayed, reserved words, comments, and C strings are highlighted in color.

Step 19: Click "main" in the [Outline] view.



The editor will jump to the line where `main()` exists and highlight it. Furthermore, a bar indicating the range of the `main()` function will be displayed in the marker bar on the left side of the editor window. This way the editor allows you to inspect functions, etc. easily.

Step 20: Double-click "boot.s" in the [C/C++ Projects] view.



Multiple sources can be opened at the same time. Click the tab at the top of the editor window (where a file name is displayed) and select the source you want to display or edit.

When assembler sources are displayed, the labels, directives, and registers are highlighted.

Step 21: Click the  (close) button on the editor tab of each open source to close the editor.

### 3.3.4 Editing the Build Options and the Linker Script

To build a project (to generate an executable object file), **make.exe** is used to start the compiler, assembler, and linker. Although the makefiles required for make are generated automatically by the **IDE**, the build options to be written in those files (i.e., compiler, assembler, and linker startup options) must first be set before they can be used. Furthermore, the contents of the linker script files required for link operation must also be set before a project can be built.

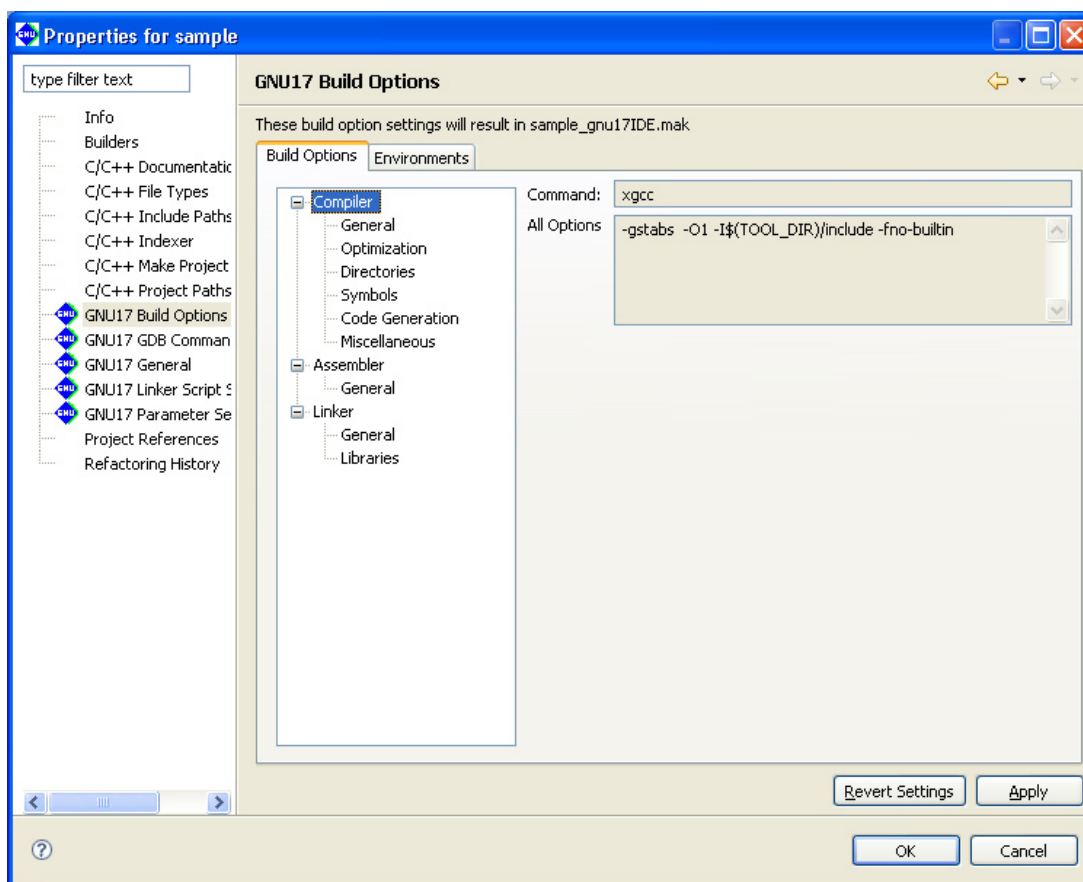
The method for making these settings is outlined below.

#### Setting build options

Step 22: Select [Properties] from the [Project] menu. You also can select [Properties] from the context menu that pops up when you right-click on the project name "sample" in the [C/C++ Projects] view.

The [Properties] dialog box will be displayed.

Step 23: From the properties list on the left side of the dialog box, select [GNU17 Build Options] by clicking on it to display the [Build Options] tab page.



Here, you can set command line options for the compiler, assembler, and linker.

When you select one of the tool names shown in tree form (Compiler, Assembler, or Linker) by clicking on it, the currently selected options are displayed in the [All Options] column. Select the kind of option from those shown in tree form by clicking on it, and the options of the selected kind will be enabled, allowing you to set. Currently displayed here are the options that have been set by default when you created a new project.

For the contents of options, refer to the respective chapters in this manual in which each tool is described. For detailed information on option select screen, refer to Section 5.7, "Building a Program".

In this tutorial, although no particular changes are needed here, we'll take a look at the method on how to add a user include path and a library file.

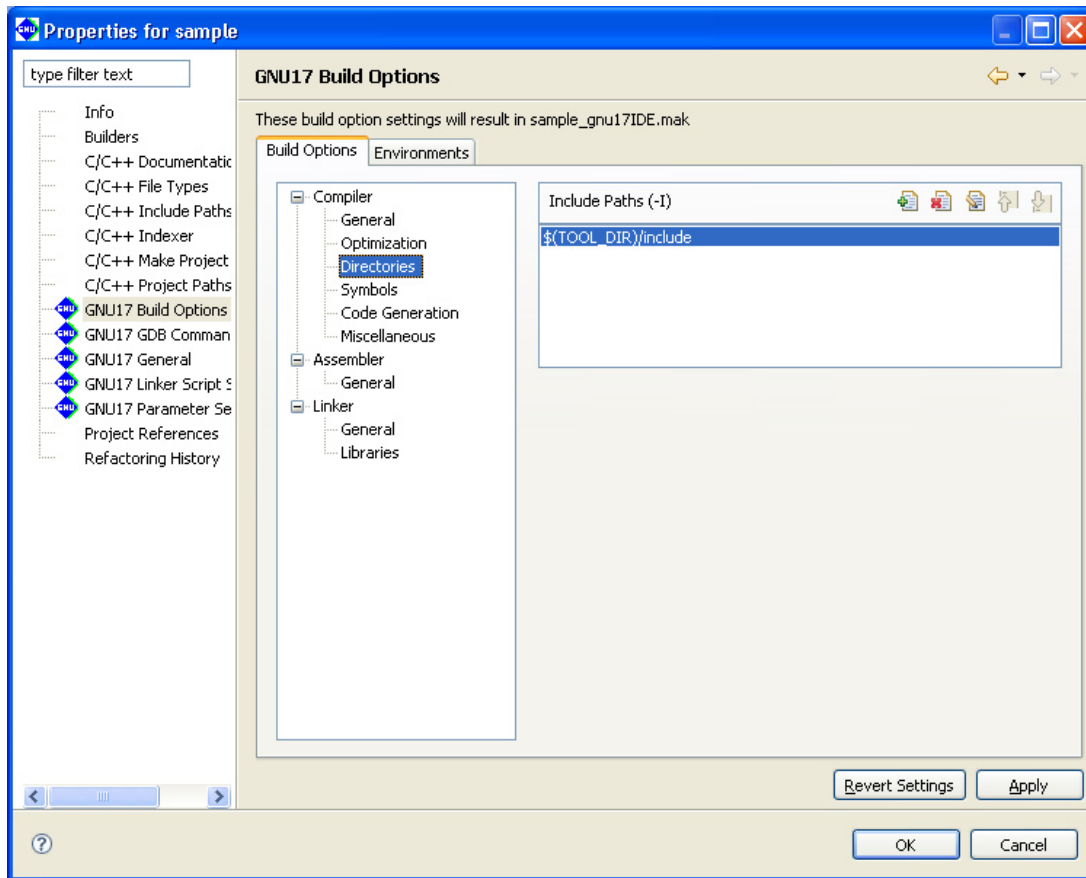


## Adding an include path


Steps 24 to 27 below are shown for reference only. No operation is required.

Step 24: Select [Compiler] > [Directories] from the [Build Options] tree.

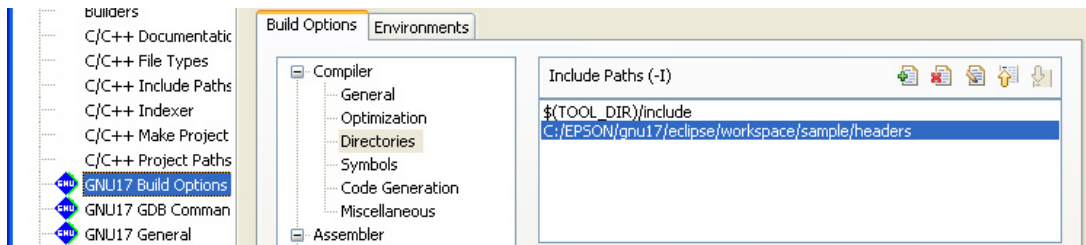
The page in which you set the C compiler's -I option (to specify an include path) will be displayed.



If user header files are prepared in another directory, they should be added to this list following the procedure described below.

 Step 25: Click the [Add] button. A directory select dialog box will be displayed, so enter a path or select one from the folder select dialog box that appears when you click the [Browse...] button.

When the directory select dialog box is closed, the path entered or selected is added to the list as shown below.





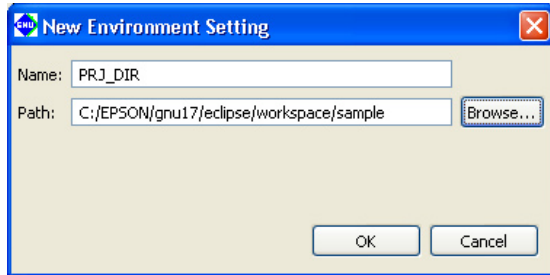
The definition procedure is described below.

**Step 27:** Click the [New] button to display the [New Environment Setting] dialog box.

Enter an environment variable name in the [Name:] text box.

Type in using the keyboard or select using the [Browse...] button to enter a path in the [Path:] text box.

Then click the [OK] button to close the dialog box.



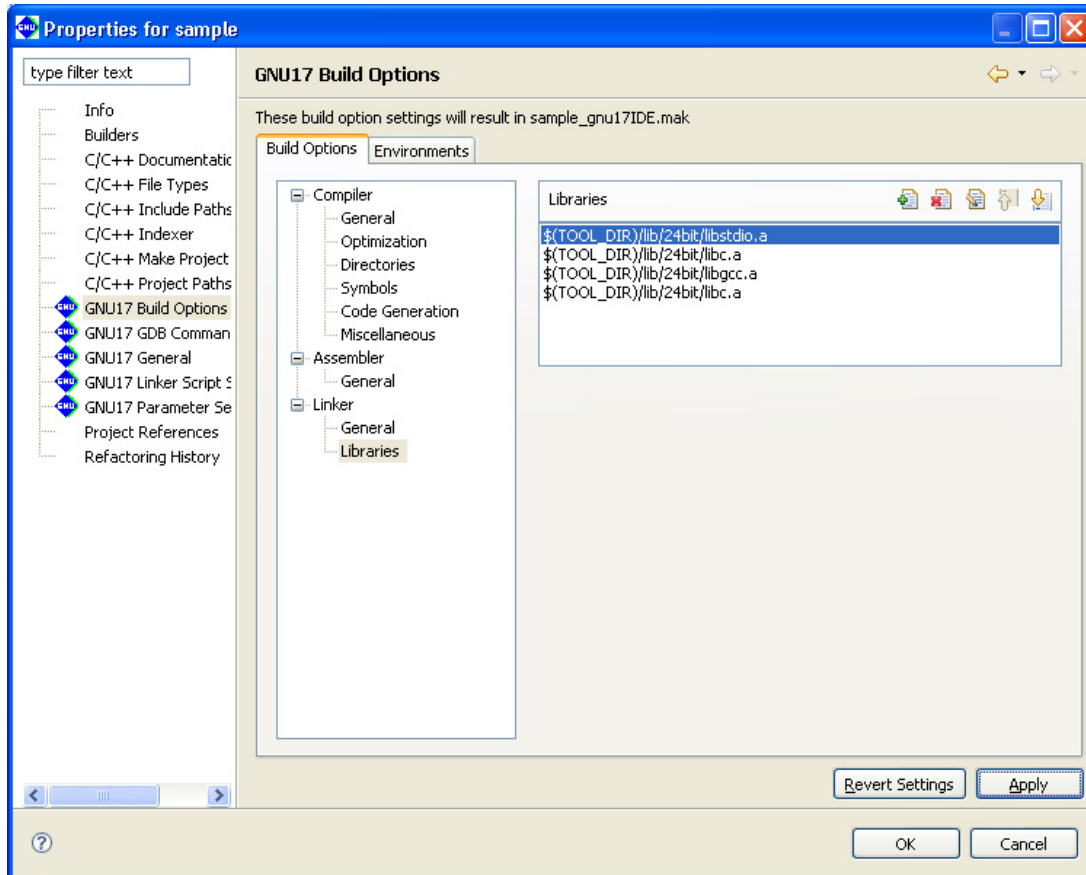
The environment variables defined here may be used for specifying include file and library file paths in the build options. The environment variable should be used as a  $\$(environment\ variable)$  macro format when specifying a path option.

### Adding a library file

Steps 28 to 30 below are shown for reference only. No operation is required.

**Step 28:** Select [Linker] > [Libraries] from the [Build Options] tree.

Displays the page in which a library file can be set.



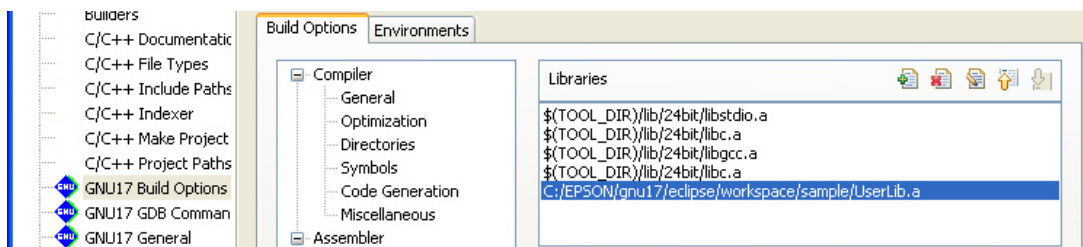
The [Libraries] column lists the ANSI library and emulation library included in this package.

If user library files are available, add them to this list following the procedure described below.



**Step 29:** Click the [Add] button. In the file select dialog box displayed, enter a file name or select one from the [Open] dialog box displayed by clicking the [Browse...] button.

Paths can be specified using the environment variables that have been defined in the [Environments] tab page. Close the file select dialog box to add the file entered or selected to the list as shown below.



The features of other buttons are the same as for the include path described before.

**Step 30:** Press the [Apply] button to confirm the changes made here.

If you click the [Apply] or [OK] button after settings in a [GNU17 Build Options] page have been changed, a dialog box appears for selecting whether the files created with the previous settings will be deleted (and rebuild) or not. Click the [Cancel] button.

The library settings specified here will be used in the linking operation.

### Setting a linker script

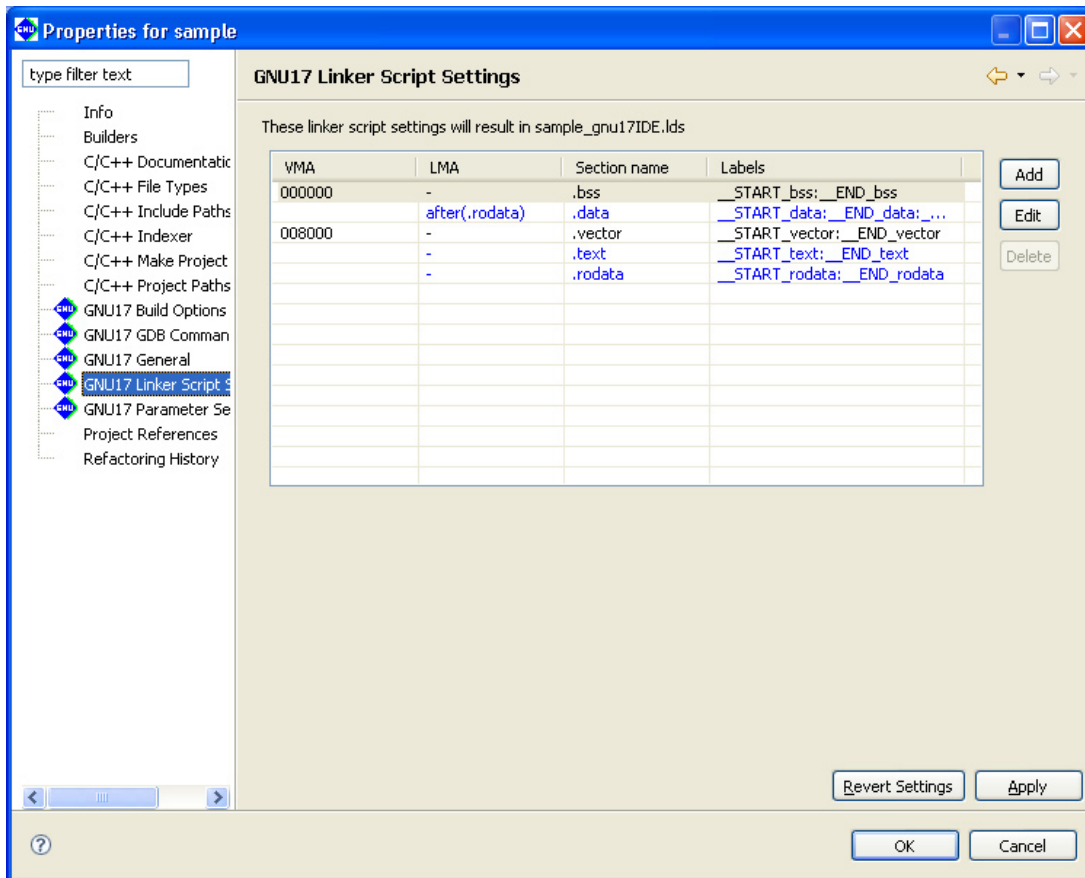
A build process requires a linker script file. This file also can be created with the **IDE**.

A linker script file is used to indicate the section location and configuration to the linker. For example, one object file generated by the assembler consists of sets of codes classified by data attributes, such as a program code part, static data part, and a variable part. A set of codes like these comprises a single section. To the linker, these represent an input section. The linker combines multiple input sections of the same kind into one (by reconfiguring them into an output section) to generate an executable object file. Furthermore, these sets of codes, even of the same attributes, must be separated by location address and device so that the program code part for the object generated from sources 1 and 2 is located at address A of the external ROM, and the program code part for the object generated from source 3 is located at address B of the internal ROM before they can be linked.

Therefore, a linker script file specifies which input sections should be combined to configure one output section, from which address a section should be stored in memory, and at which address a section should be executed. For more information, refer to Section 3.8, "Sections and Linkage".

**Step 31:** If you closed the [Properties] dialog box, select [Properties] from the [Project] menu to reopen it.

**Step 32:** Click to select [GNU17 Linker Script Settings] from the properties list.



You'll see that the five basic sections (output sections)—i.e., `.bss`, `.data`, `.vector`, `.text`, and `.rodata`—are preset in the Section name column.

<code>.bss</code>	Section in which variables without initial values are placed. (Normally located in RAM.)
<code>.data</code>	Section in which variables with initial values are placed. (The initial values are located in ROM. They are copied into RAM when needed.)
<code>.vector</code>	Section in which vector tables are placed. (The actual data is located in ROM.)
<code>.text</code>	Section in which program codes are placed. (The actual data is located in ROM and executed there or from high-speed RAM after copying.)
<code>.rodata</code>	Constant variables. (The actual data is located in ROM.)

The VMA (Virtual Memory Address) is the position (start address) at which a section is placed during runtime. If a section does not have its start address indicated in the VMA column, it means that the section is to be located following the immediately preceding section.

The LMA (Load Memory Address) is the position (start address) in ROM at which the actual data is placed. If this column is marked with "-", it means that this address is the same as the VMA (i.e., the section will be executed or accessed at the position at which the actual data is placed). If this column is marked with "after (.rodata)", it means that the actual data is to be located following the section indicated in parentheses (in this case, the `.rodata` section).

The Labels column shows the labels indicating the start and end addresses of an area in which the section will be located. If the LMA is not specified, two labels *<beginning of VMA>* and *<end of VMA>* are shown here. If the LMA is specified, four labels are shown in order of *<beginning of VMA>*, *<end of VMA>*, *<beginning of LMA>*, and *<end of LMA>*. These labels may be used to specify addresses in a source file when, for example, copying sections from ROM to RAM.

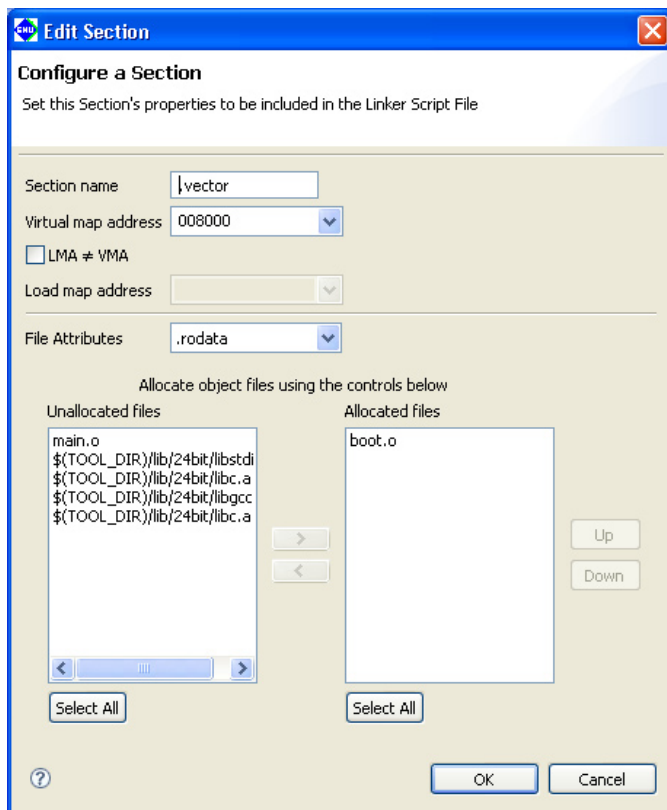
During actual program development, user defined sections can be added using the [Add] button.

The section information is displayed in blue except for the `.vector` section displayed in black. Blue is used to display the standard sections defined by default and black is used to display other user defined sections. To edit the section name, standard section attribute, address to locate, and objects to be located, a user section should be created. The standard section allows the user to specify the location address only, and objects are automatically located except those are located in the user sections with the same attribute.

Each of the above sections is predefined to contain object files that exist within a project. Let's take a look at an example of the `.vector` section.

**Step 33:** Select "`.vector`" from the section list and click the [Edit] button.

The [Edit Section] dialog box will be displayed.



The upper part of the dialog box is used to set the sections listed in the preceding screen.

The list box on the lower right side shows the objects to be located in the `.vector` section. You can see "`boot.o`" is set in the `.vector` section as you have previously specified in the New Project Wizard.

The list box on the left side lists the remaining other object files and library files within the project.

If any object in the left-side list needs to be located in this section, select that file from the list and click the [>] button. The selected file will be moved to the right-side list box and added to the list of files that comprise this section.

If there are multiple files displayed in the right-side list, they will be located in order as shown. The placement order can be changed with the [Up] or [Down] button.

Although there are no object files generated at this point of time yet, the files in this dialog box are displayed on the assumption that object files (`boot.o`, `main.o`) will be generated from the source files added to a project in the same name as those of the source files.

To take a look at [File Attributes] here, we see that the indicated attribute is ".rodata". This means that only the .rodata sections in boot.o will be located in the .vector section. Since the other sections in boot.o will be located in respective sections with the same attribute, looking at the other section information we find that all sections except the .rodata section with the same attribute will have boot.o located in each.

In its initial settings, the **IDE** assumes that a vector table is written in the .rodata section (in the C sources, the constants declared by `const`, in the assembler sources, the constants in the scope of the .rodata section).

If the vector table is written in another section with a different attribute (e.g. .text section), select the attribute from [File Attributes] so that the section will be located in the .vector section.

Furthermore, [Virtual map address] contains the boot vector address specified when the project is newly created. If the processor has a different boot vector address, rewrite [Virtual map address] with the correct value.

**Step 34:** Click the [Cancel] button to close the dialog box.

**Step 35:** Click the [OK] button in the [Properties] dialog box to finish editing a linker script.

Editing objects to be located and section attribute of a user section (displayed in black) automatically updates the object configuration of the standard section with the same attribute.

By the above, you are finished with preparations for building a program.

### 3.3.5 Building a Program

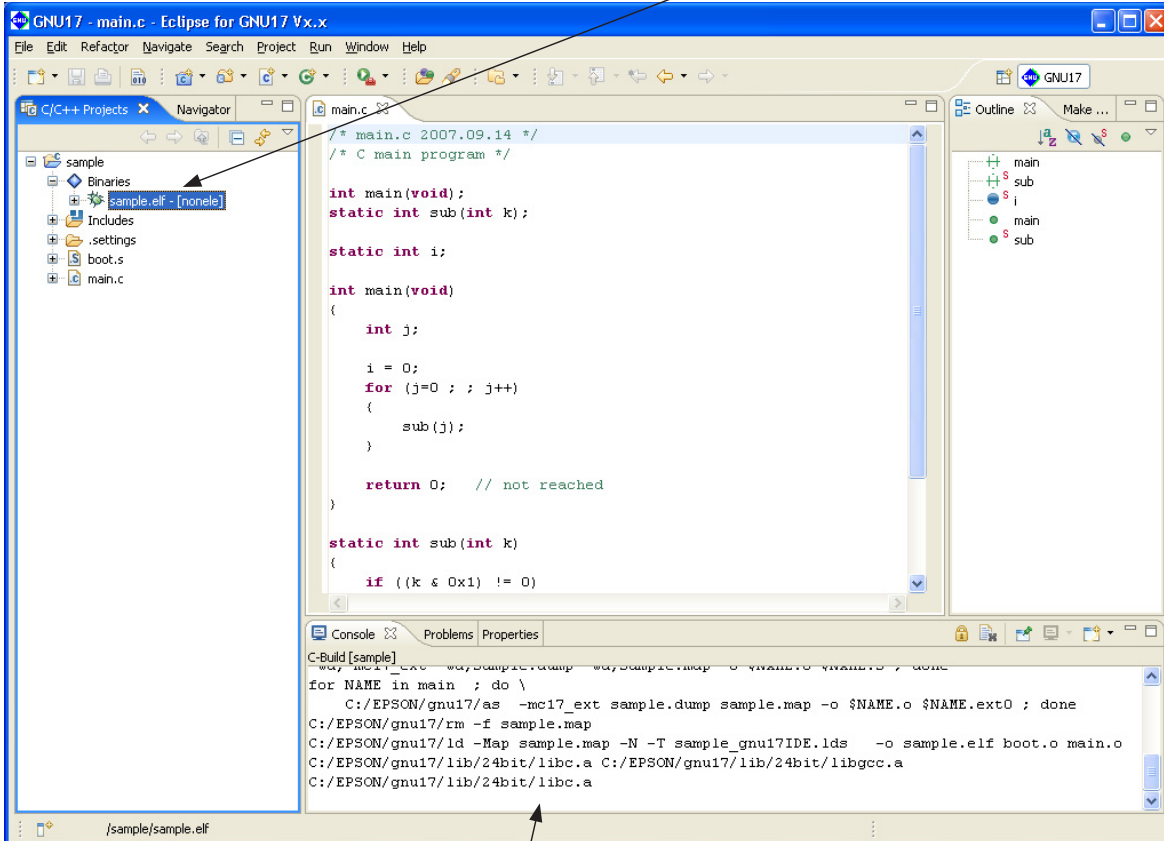
When you are finished with the work described in the preceding sections, you are ready to build (compile, assemble, and link) a program.

#### To execute a build process

Step 36: Select the project name "sample" from the [C/C++ Projects] view.

Step 37: Select [Build Project] from the [Project] menu. You also can select [Build Project] from the context menu that appears when you right-click on the project name "sample" in the [C/C++ Projects] view.

When the build command is selected this way, makefiles are generated with the current settings and then **make.exe** is executed to generate an executable format object file `sample.elf`.



The commands executed during a build process and tool messages are displayed in the [Console] view.



### 3.3.6 Debugging a Program

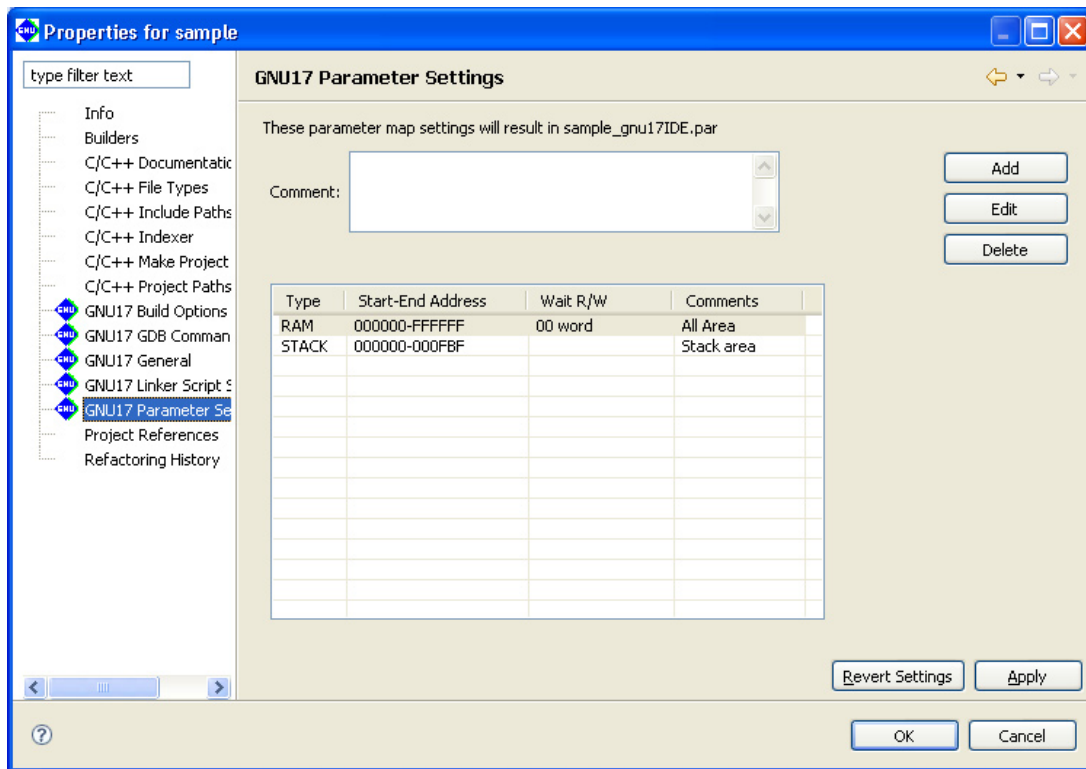
Before debugging a program, create a parameter file for the debugger. This is a file in which the memory map information of the target system is written, which is loaded into the debugger to set a memory map. A parameter file should be created to be suitable for the memory configuration of the target system and must always be loaded into the debugger.

Furthermore, the debugger's startup options must also be set before debugging a program.

#### Setting parameters

Step 38: Select the "sample" project in the [Navigator] or the [C/C++ Projects] view and then [Properties] from the [Project] menu.

Step 39: Select [GNU17 Parameter Settings] from the properties list by clicking on it.



Two items of area information that have been set by default will be displayed.

The information for RAM defines that 0x0 to 0xfffff (16M bytes) be used as a RAM area. Note that "00 word" here means this device is accessed in 32-bit size for read with no wait states (0 cycles) and for write with no wait states (0 cycles). (The access conditions set here are effective in only simulator mode.)

Other area information for the stack area in RAM is also defined.

Shown here is the basic configuration of the S1C17 microcomputer that incorporates the S1C17 Core.

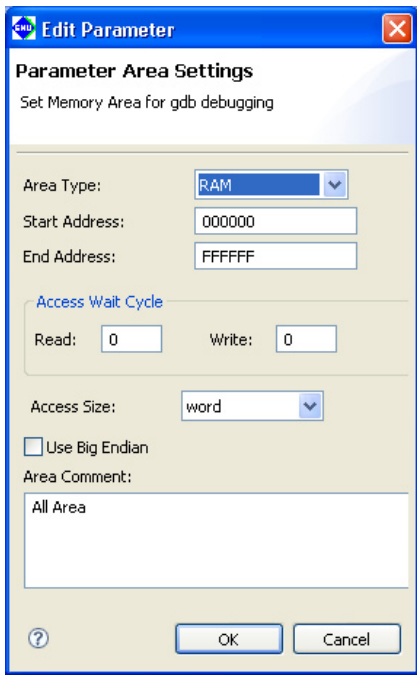
If other memory or external devices must be used, click the [Add] button and set the area to be added.

Since the sample program does not specifically require a memory configuration other than the default, a parameter file may be created directly as shown here without incurring any problem. Steps 40 to 46 below are shown for reference only. No operation is required.

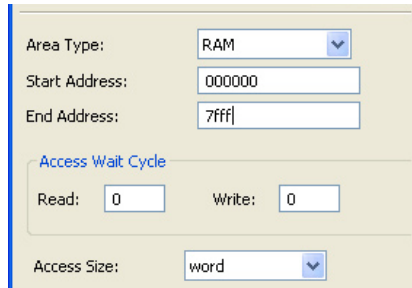
As for an example, we'll add a ROM area (0x8000–0x8fff). Note, however, that areas cannot overlap with another area except for the STACK setting. Therefore, first the RAM area must be changed to 0x0–0x7fff.

Step 40: Click on the RAM line in the list box to get it displayed in inverse video and click the [Edit] button.

The [Edit Parameter] dialog box will be displayed.



Step 41: Change the address in [End Address:] to 7fff.

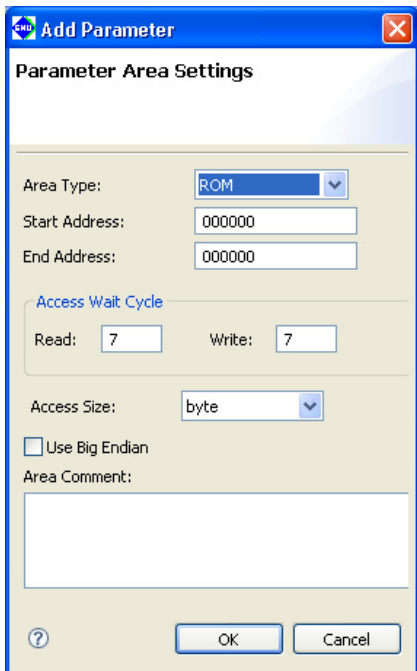


Step 42: Click the [OK] button.

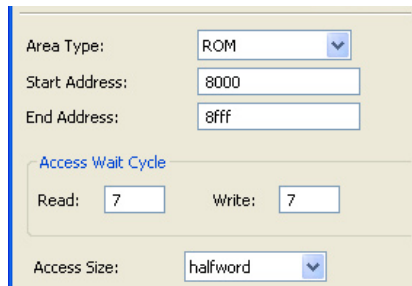
The displayed address range of the RAM area has been changed to "000000-007FFF".

Step 43: Click the [Add] button.

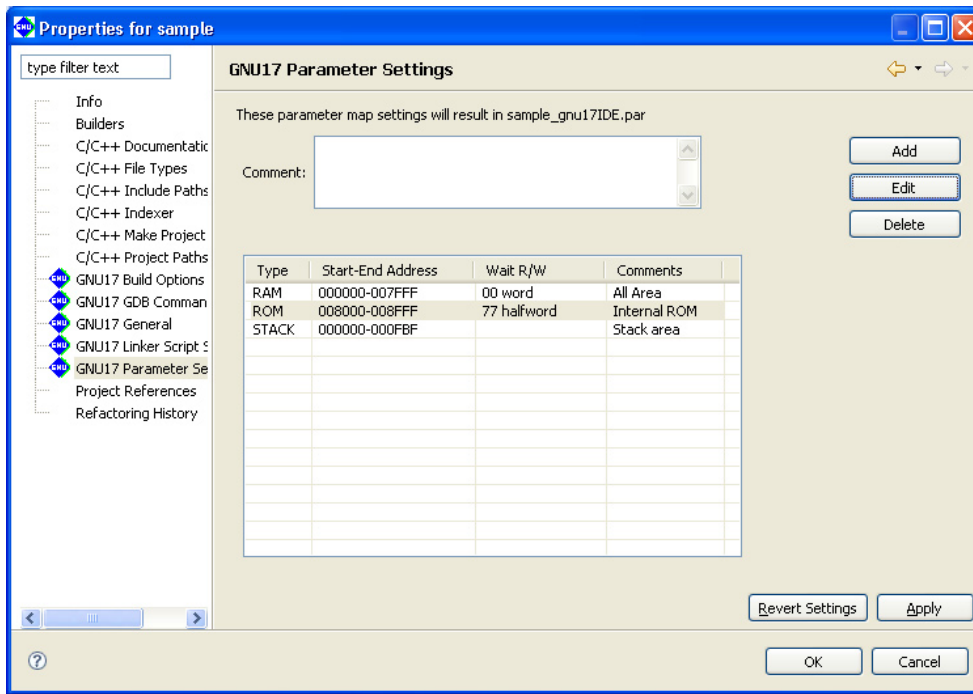
The [Add Parameter] dialog box will be displayed.



Step 44: Enter 8000 in the [Start Address:] text box and 8fff in the [End Address:] text box. Select "halfword" (16 bits) from [Access Size:].



Step 45: Click the [OK] button.



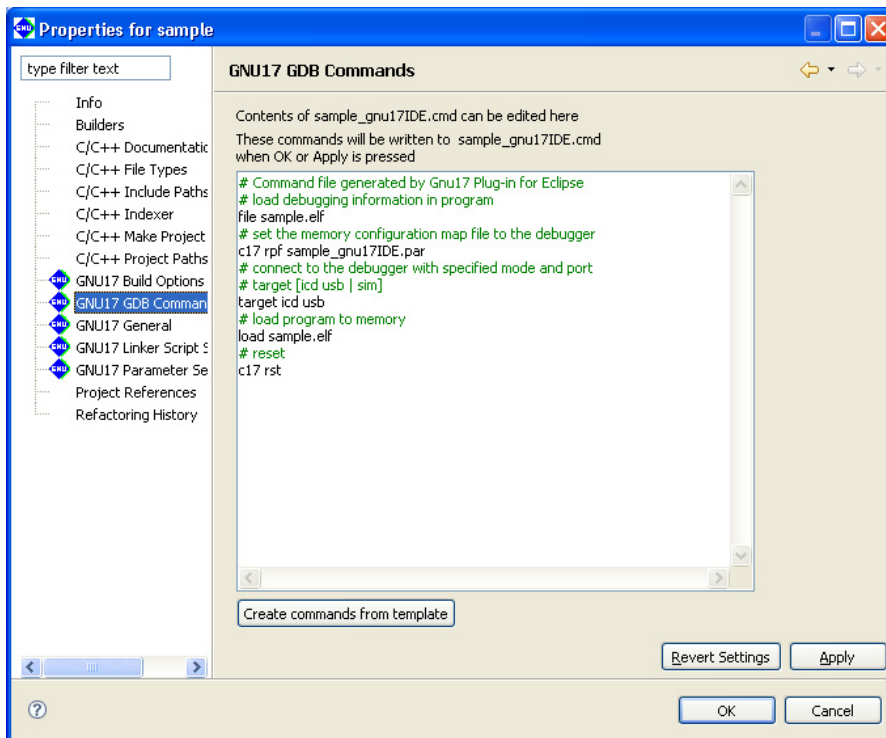
The ROM area have been added.

Step 46: Click the [Apply] button.

When above settings are made, a file named "sample\_gnu17IDE.par" is generated and passed to the debugger via a command file when the debugger starts.

### Setting the debugger's startup options

Step 47: Select [GNU17 GDB Commands] from the properties list by clicking on it.



This page displays the contents of the debugger startup command file that will be generated by the IDE.

The debugger must be set to the appropriate mode that suits the ICD used, etc. before it can be operated. For detailed information, refer to Section 3.7, "Debugging Environment".

### 3 SOFTWARE DEVELOPMENT PROCEDURES

Here, we'll set the debugger to simulator mode that does not require external equipment before we start debugging.

The contents of the command file displayed by default are provided for debugging using an ICD.

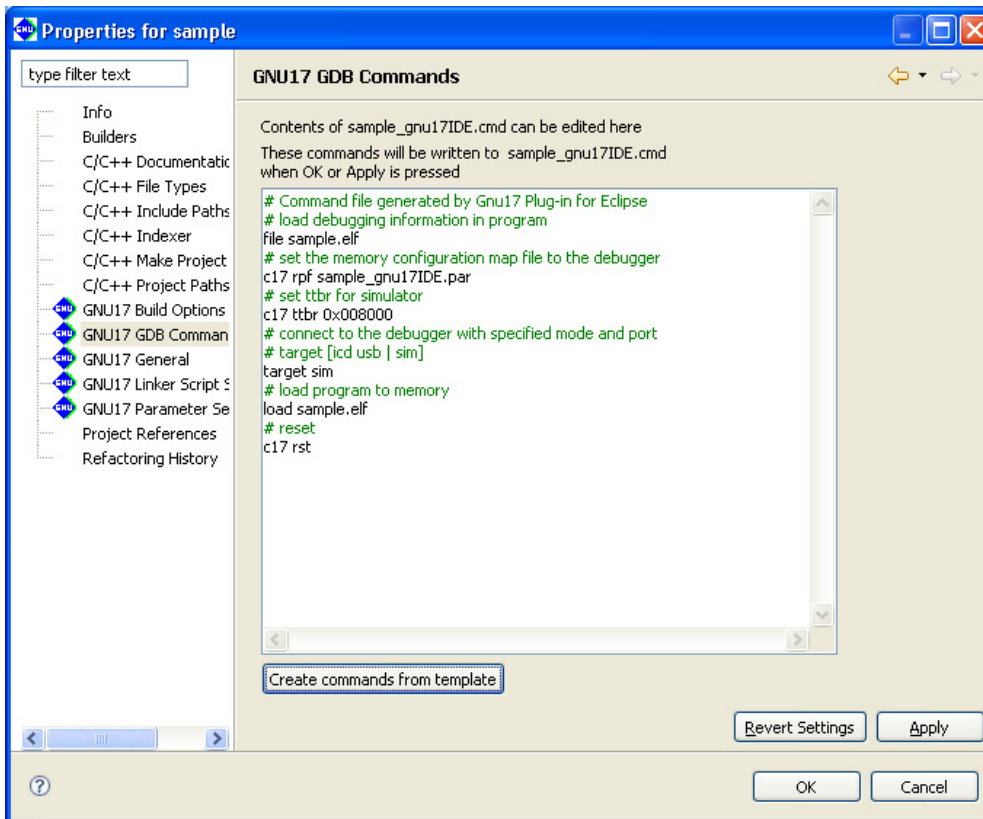
It may be changed for simulator mode by the following procedure.

**Step 48:** Click the [Create commands from template] button to display the [Create a simple startup command] dialog box.

**Step 49:** Select "Simulator" from the [Debugger:] combo box.



**Step 50:** Click the [Overwrite] button.



The displayed contents are altered for simulator mode. The commands may be added and edited directly in this page as necessary.

**Step 51:** Click the [OK] button.

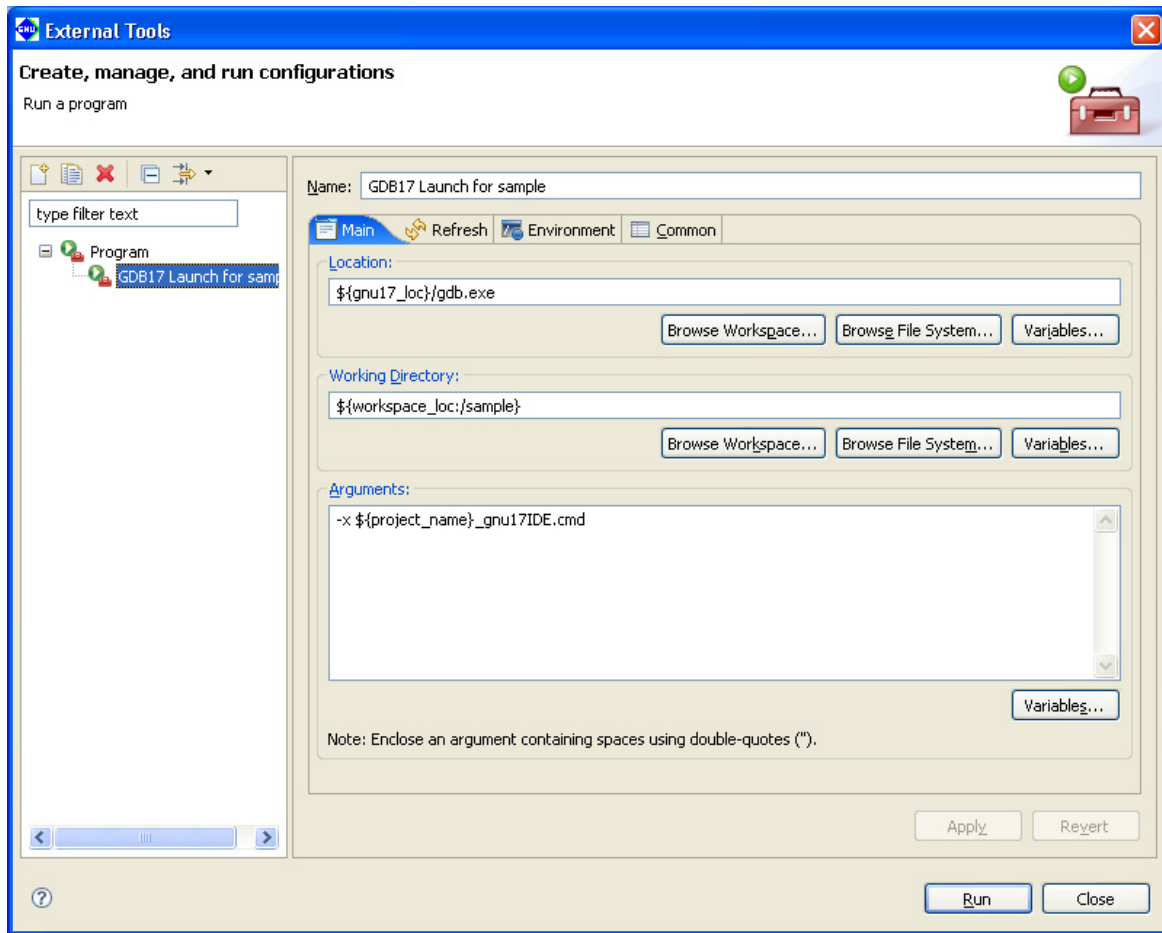
When above settings are made, a command file named "sample\_gnu17IDE.cmd" is generated and it will be passed to the debugger.

## Starting the debugger

Step 52: Select [External Tools] > [External Tools...] from the [Run] menu.

The [External Tools] dialog box will be displayed.

Step 53: Select [GDB17 Launch for sample] from the list in the dialog box.

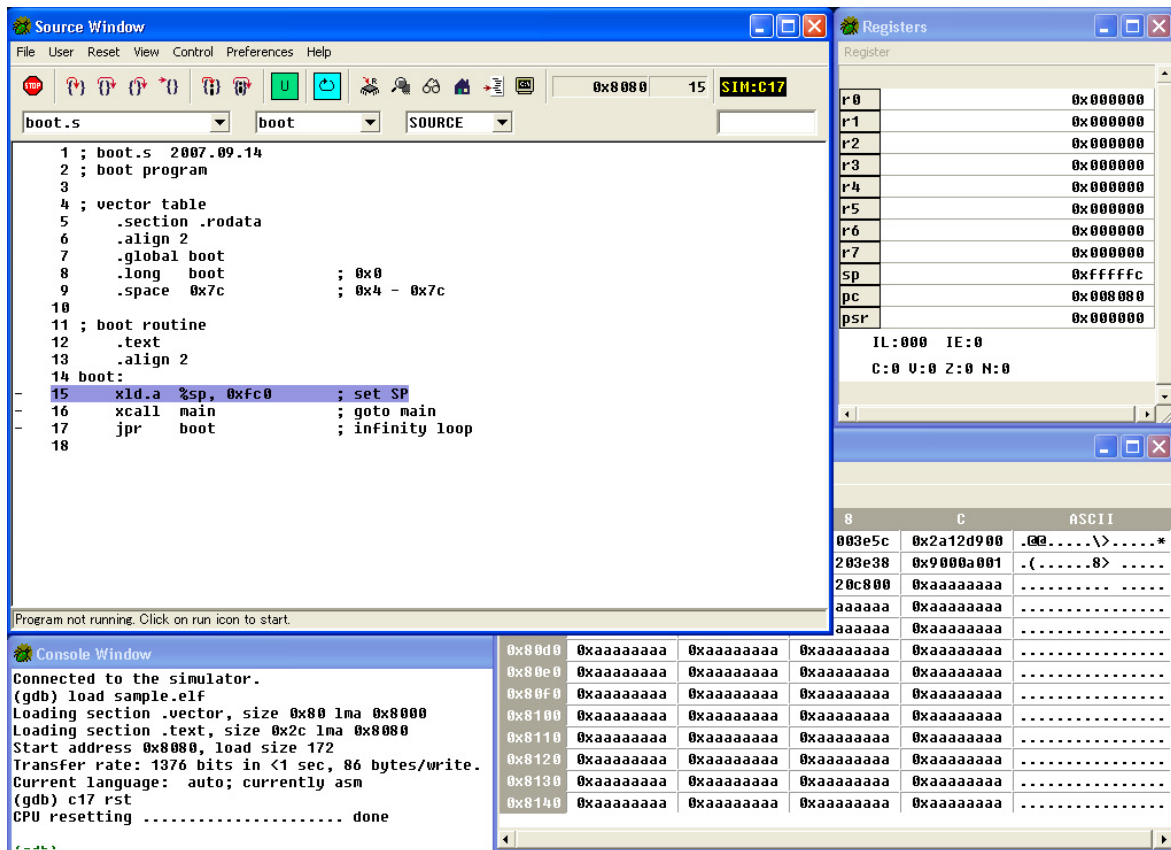


This dialog box may be used to edit the command line of the **gdb** debugger. No particular changes are required for executing the sample, so start the debugger directly with this setting.

Step 54: Click the [Run] button.

The **gdb** debugger will start.

When the debugger has started, the window shown below appears, executing the command file that was set in the [GNU17 GDB Commands] dialog box.



The object file is loaded into the debugger by the command file and the debugger is reset. The PC (program counter) is set to the program execution start position, letting the debugger ready to start debugging.

### To run a program



Step 55: Click the [Continue] button in the toolbar.

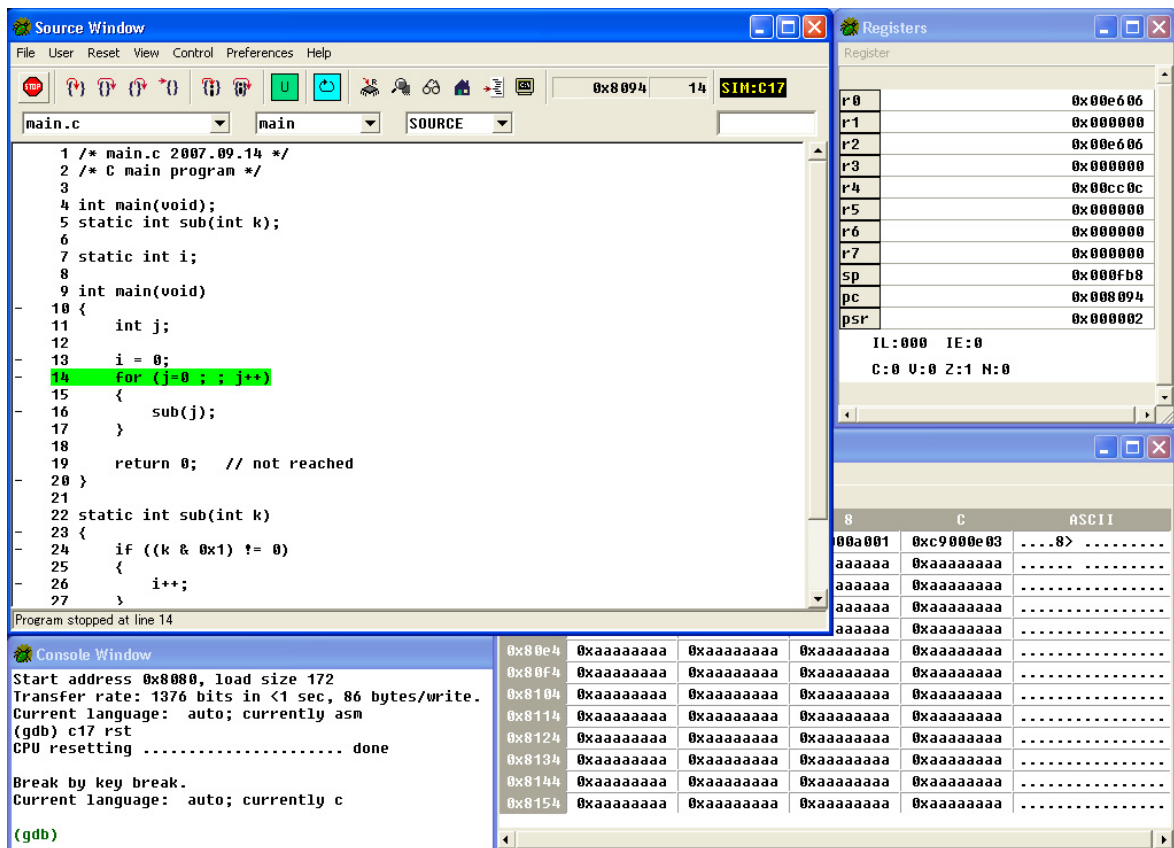
The sample program here endlessly increments the int variable counter 'i' (addresses 0x0–0x3).

Use a forcible break to stop such an endless loop.

### To forcibly break a program



Step 56: Click the [Stop] button in the toolbar.



Notice that the `for` statement in the [Source] window is highlighted in green. This is because the current address of the PC (program counter) exists there, at which the program has stopped. Furthermore, notice that the 'pc' column of the [Registers] window indicates the address `0x8094`. It means that the program has stopped immediately before executing the instruction at this address.

Although `boot.s` was displayed in the [Source] window when the program has started, the source of `main.c` is displayed in it because the program has stopped and remains idle in `main.c` now.

The [Source] window can display a program in other than the source mode.

### To change the display mode of the [Source] window

There is the pulldown list box that shows `SOURCE` in the toolbar. Use it to specify the display mode of the [Source] window.



Step 57: Select "SOURCE", "ASSEMBLY", "MIXED", or "SRC + ASM" in the pulldown list to change display modes.

## SOURCE

In SOURCE or the default display mode, the content of a source file is displayed beginning with the top of the file, with line numbers added. You can see the entire source file in one window.

```

1 /* main.c 2007.09.14 */
2 /* C main program */
3
4 int main(void);
5 static int sub(int k);
6
7 static int i;
8
9 int main(void)
10 {
11     int j;
12
13     i = 0;
14     for (j=0 ; ; j++)
15     {
16         sub(j);
17     }
18
19     return 0; // not reached
20 }
21
22 static int sub(int k)
23 {
24     if ((k & 0x1) != 0)
25     {
26         i++;
27     }

```

Program stopped at line 14

## ASSEMBLY

In ASSEMBLY mode, the loaded object code is displayed in disassembled form. Even when the source files corresponding to the loaded object file cannot be found, the object code is displayed in this mode. The C sources are displayed in function units. Only the currently halted function can be displayed at a time, and no other functions can be displayed in the same window.

```

0x8088 <main>:      ld.a    -[%sp],%r4      ld.a    -[%sp],%r4
0x808a <main+2>:    ld      %r2,0x0        ld      %r2,0x0 <i>
0x808c <main+4>:    ld      [0x0],%r2      ld      [0x0],%r2 <i>
0x808e <main+6>:    ld      %r4,%r2        ld      %r4,%r2
0x8090 <main+8>:    ld      %r0,%r4        ld      %r0,%r4
0x8092 <main+10>:   call    0x4            call    0x4    (0x
0x8094 <main+12>:   add     %r4,0x1        add     %r4,0x1
0x8096 <main+14>:   jpr     0x3fc          jpr     0x3fc  (0x
0x8098 <main+16>:   ld.a    %r4,[%sp]+    ld.a    %r4,[%sp]+
0x809a <main+18>:   ret

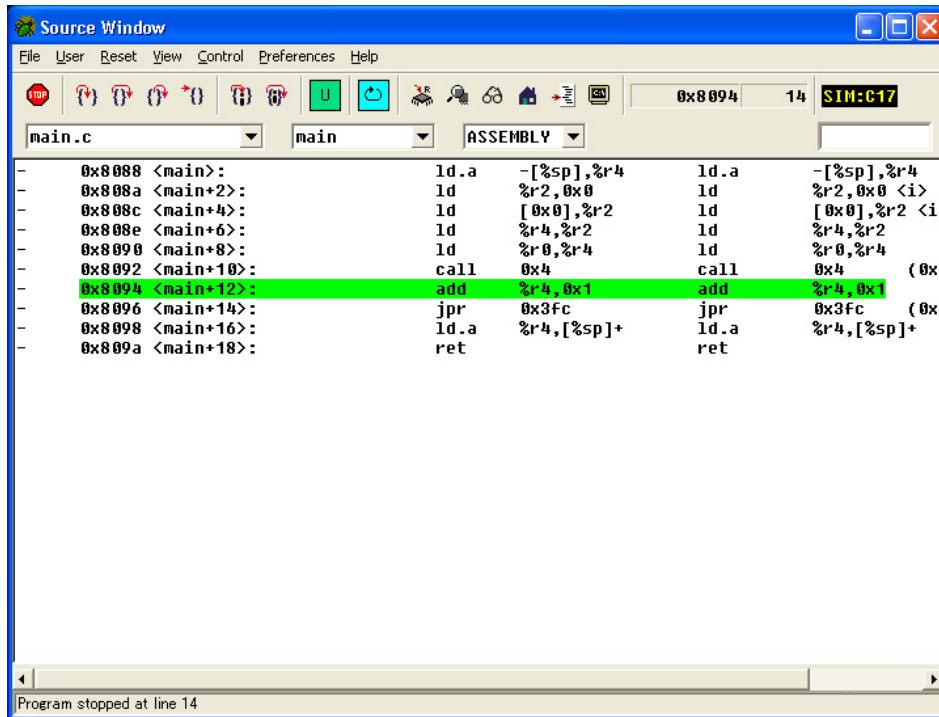
```

Program stopped at line 14

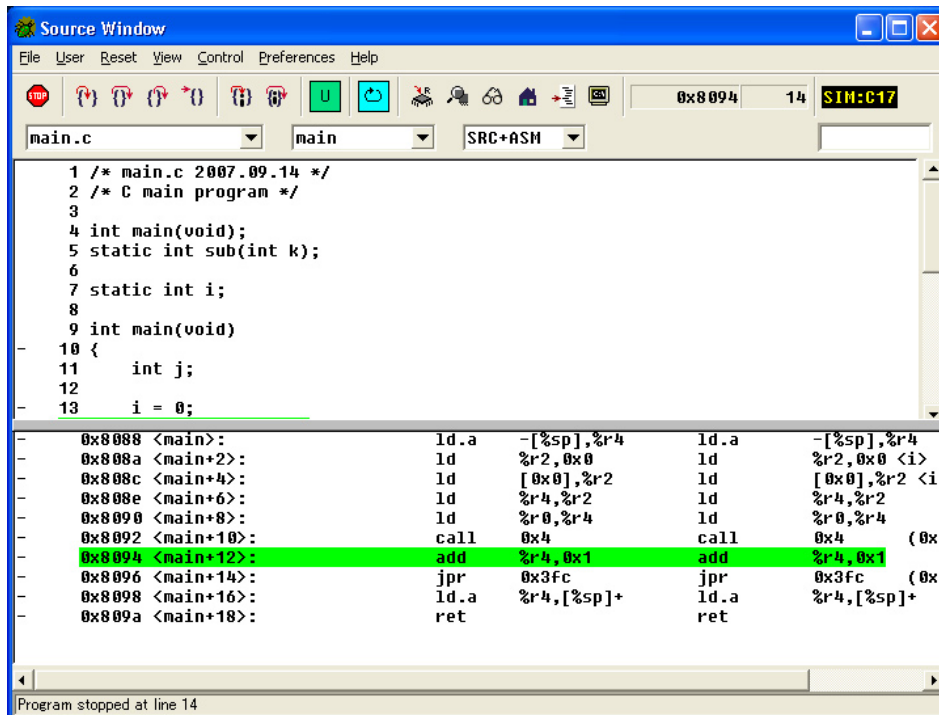


**MIXED**

In MIXED mode, the displayed source lines have the corresponding assembler display inserted in each. This MIXED mode, therefore, allows you to know not only the source line at which the program has stopped, but also the address and the instruction code at that address, all in the [Source] window. In this case too, the C sources are displayed in function units.

**SRC+ASM**

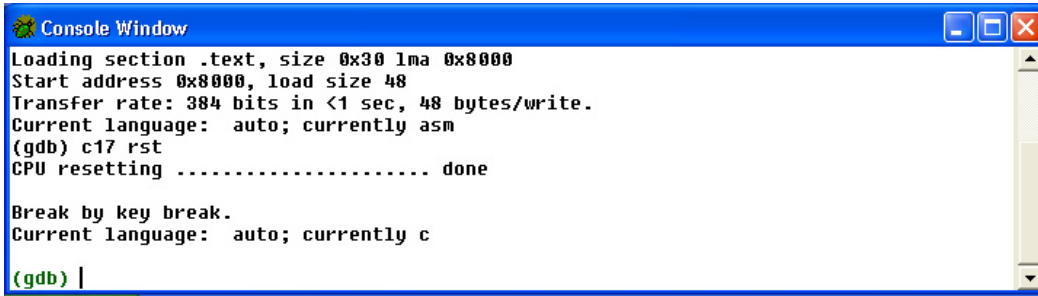
In SRC + ASM mode, the window is split into the upper and lower parts for simultaneous display in SOURCE and ASSEMBLY modes.



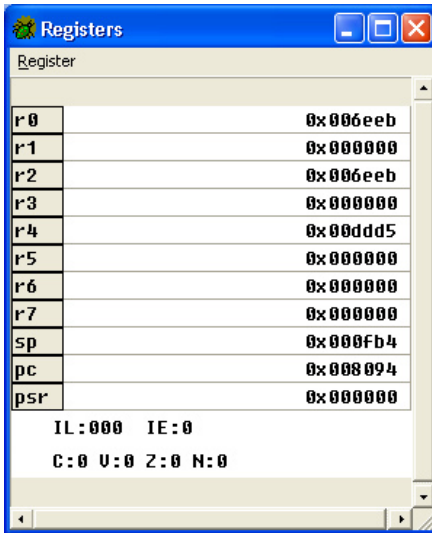
Step 58: Select SOURCE to reverse the display mode.

Let's take a look at other windows of the debugger here.

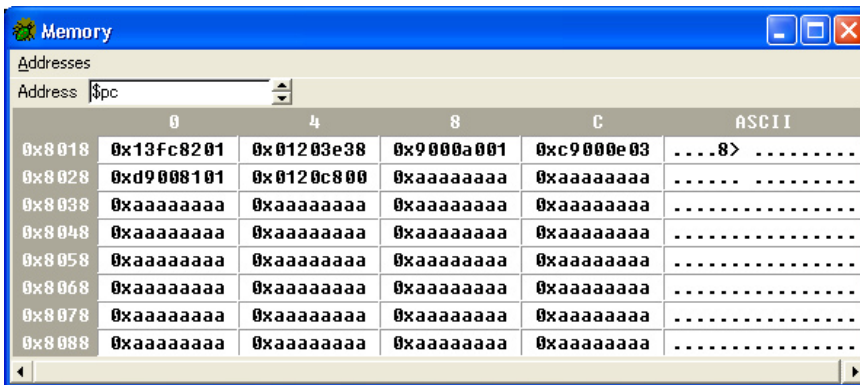
### About the debugger windows



This is the [Console] window. Enter a debug command at the " (gdb) " prompt to execute it. Although we used a button to run a program in Step 55, the same effect can be achieved by entering `cont` here and pressing the [Enter] key.



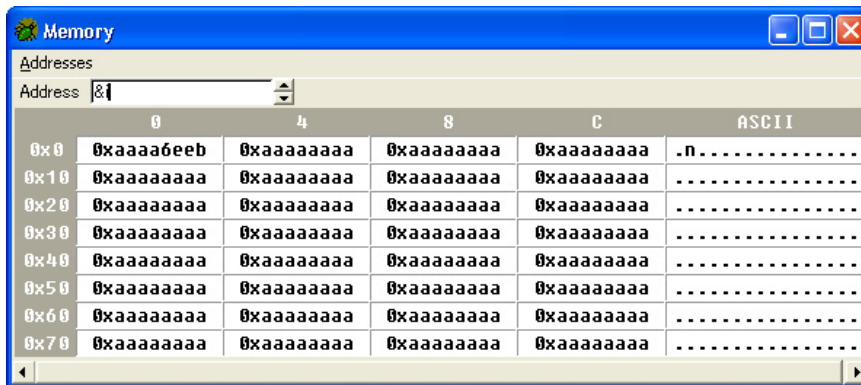
This is the [Registers] window. It shows the contents of the S1C17 Core registers. The register data can be rewritten here.



This is the [Memory] window. It shows the contents of the target memory. The memory data here can be rewritten.

When run in the above step, the sample program increments the int variable 'i' (addresses 0x0–0x3). Examine value of the variable 'i' in the [Memory] window.

Step 59: Enter '&i' or '0' in the [Address] text box of the [Memory] window and press the [Enter] key.

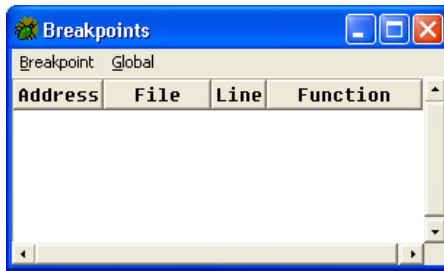


The memory contents are displayed, beginning with the variable 'i' (address 0x0). As shown here, 'i' has been counted up to 0x6eeb (= 28395).

These three windows are displayed by default when the debugger starts, along with the [Source] window. In addition, the following three windows are available.



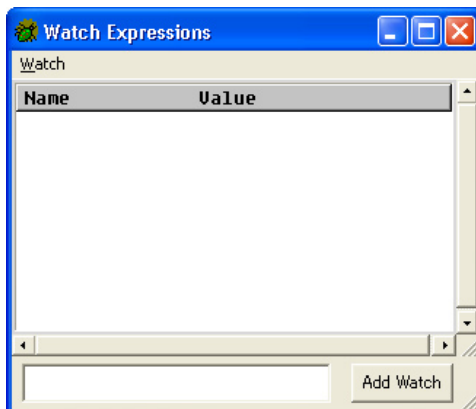
Step 60: Click the [Breakpoints] button.



This is the [Breakpoints] window. This window is used to manage software PC breakpoints that halt the program at specified positions.

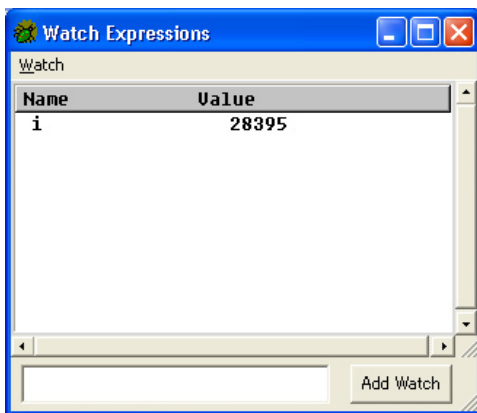


Step 61: Click the [Watch Expressions] button.



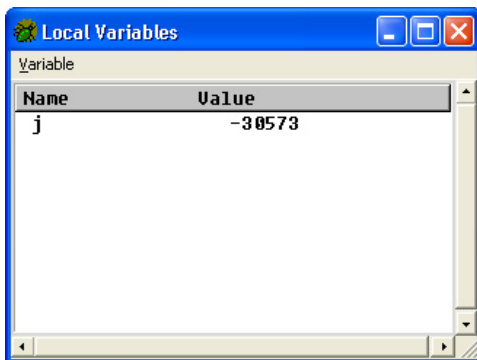
This is the [Watch Expressions] window. This window is used to monitor the values of global variables. This window may be used to monitor the variable 'i' (i = global variable) earlier verified in the [Memory] window. The procedure is described below.

Step 62: Enter 'i' in the text box located at the bottom of the window and click the [Add Watch] button.



The letter 'i' will appear in the list box of the window, the contents of which are displayed as decimal values (default display mode).

 Step 63: Click the [Local Variables] button.



This is the [Local Variables] window. It shows the local variables defined in the current function. Since the current PC address exists in the `main()` function, the symbol and the value of the variable 'j' defined in this function are displayed.


We have thus far seen the windows for the **gdb** debugger. Each window has other facilities, not just the ones that display information. These are detailed in Section 10.4, "Windows".



We'll now return to program execution.

In the preceding steps, we ran a program, stopping it using forced breaks.

This time we'll run a program after specifying in advance a position at which to stop it.

### To specify a breakpoint


Step 64: The source line numbers are displayed in the [Source] window. Move the mouse cursor to a position preceding numeral 16. Click when the cursor changes to a  (white circle).

You will see that source line 16 is marked with  (red) at the beginning of it. This means that this line has been set to be a software PC breakpoint. If a  mark is attached anywhere other than source line 16, click there to reverse, then repeat.

```

- 13   i = 0;
- 14   For (j=0 ; ; j++)
- 15   {
- 16   sub(j);
- 17   }

```

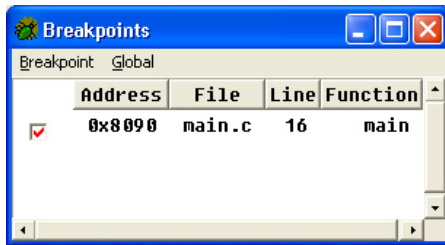
 Step 65: Click the [Continue] button.

In contrast to Step 55, the program this time should have stopped at the line set to be a breakpoint. Try pressing the [Continue] button a number of times. You will see that the program stops at the same place each time.

If the [Local Variables] window is still open, you can verify that the variable 'j' increments each time the program breaks. Similarly, you can verify that the variable 'i' displayed in the [Watch Expressions] window increments every other time the program is run, indicating that the program operates exactly as expected.



Step 66: Click the [Breakpoints] button.



Examine the [Breakpoints] window. The information on the breakpoint we set above is displayed in it, although no information was displayed there earlier. The check mark shown at the beginning of the information means that the breakpoint is currently active. When the check box is unselected, the breakpoint is temporarily disabled: The next time the program is run, it will no longer halt at the position at which it halted earlier. Selecting the check box reenables the breakpoint.

Step 67: Click on the ■ mark that is displayed in the [Source] window to turn it off.

This clears the breakpoint.

When the ■ mark disappears, the line is marked with "-" back again. This "-" mark means that the line can be set to be a breakpoint. The lines lacking this mark are source lines not converted to actual executable instructions when the source was compiled and assembled. No breakpoints can be set in these lines.

In addition, other break facilities are available, including a temporary break effective only once the program is run. Discussions of these break facilities are omitted here. For detailed information on break facilities, refer to Section 10.6.6, "Break Functions".

If any problem in program behavior is detected, the program operation should be verified with greater care. As the last step of the tutorial, we will proceed through the program by executing one source line at a time.

### To proceed through the program step-by-step



Step 68: Click the [Step] button in the toolbar.

The source line highlighted in green in the [Source] window (the line at which the current PC address exists) is executed, and the highlighting moves to the next source line to be executed.

By repeating Step 68, we can execute the program one step or one source line at a time. If the program has no problems, you will see that the displayed register values, etc. change correctly at each step.

The [Step] button executes the program one source line at a time. To execute the program one instruction (mnemonic) at a time, use the [Step Asm Inst] button.



[Step Asm Inst] button



Step 69: Click the [Next] button in the toolbar.

Repeat Step 69 to verify differences between this and the [Step] button in the [Source] window.

When the program is run with the [Next] button, you will see that although the function `sub()` was skipped, the value of the variable 'i' is updated, indicating that the instructions in the function have all been executed.

The [Next] button operates in basically the same way as the [Step] button, except that the [Next] button skips functions and subroutines (i.e., executes a function or subroutine as one step, without stopping at every instruction). If you do not need to debug the subroutines instruction by instruction, use the [Next] button instead.

### 3 SOFTWARE DEVELOPMENT PROCEDURES

We have thus far seen the basic use of the debugger. More advanced debugging can be performed by entering a command in the [Console] window from the keyboard. For detailed information, refer to Chapter 10, "Debugger".

Follow the procedure described below to quit the debugger.

#### To quit the debugger

Step 70: Select [Exit] from the debugger's [File] menu.

All of the debugger windows will be closed, and the **IDE** window is displayed once again.

In addition to the simulator mode described above, a program can be debugged in another mode after connecting an ICD to the target board. For detailed information on how to debug in this mode, refer to Section 3.7, "Debugging Environment".

Finally, we'll quit the **IDE**.

#### To quit the IDE

Step 71: Select [Exit] from the **IDE**'s [File] menu.

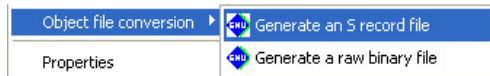
### 3.3.7 Creating ROM Data

Creating data for external ROM or masking data for internal ROM to be incorporated into the target board requires HEX files. After a program is completed, you must convert the elf format object file into a Motorola S3 format HEX file, then separate the converted file between those used for external ROM and those used for internal ROM. To do this, use the object file format conversion utility **objcopy.exe** provided standard with gnu. For detailed information on **objcopy.exe**, refer to the documents on the gnu utilities or Section 11.4 in this manual.

In addition to this, although an option and output file name cannot be specified, the IDE's [C/C++ Projects] and [Navigator] views support file conversion into Motorola S3 format through the context menu.

#### To create a HEX file on the IDE

**Step 72:** Right-click the elf format object file displayed in the [C/C++ Projects] or [Navigator] view to show a context menu, and then select [Object file conversion] > [Generate an S record file] from the menu.



This generates a Motorola S3 format file with the same name as the elf format file and ".sa" file extension.

#### To create a HEX file using objcopy

Open the command prompt window and execute **objcopy** in the command line shown below.

```
C:\EPSON\gnu17>objcopy -O srec -R .gbss --srec-forceS3 InputFile OutputFile
```

-O srec: Select Motorola format for the record format of the output file.  
 -R .gbss: This option specifies that the sections named ".gbss" should not be included in the output file.  
 --srec-forceS3: Specify that the file be output in Motorola S3 format.  
 InputFile: Specify the elf format object file to be converted.  
 OutputFile: Specify the name of the HEX file to be output.

Example: To extract all HEX data from input.elf and write it out to output.sa

```
C:\EPSON\gnu17>objcopy -O srec input.elf output.sa
```

#### Creating the mask data to be presented

After a program for a type of processor with built-in ROM is completed, you are then requested to present the masked data for the internal ROM to Seiko Epson.

After creating a Motorola S3 format HEX file with **objcopy.exe**, confirm that the blank addresses in it are filled with 0xff data using **moto2ff.exe**. For detailed information on **moto2ff.exe**, refer to the sections in which other tools are described.

The following describes how to create ROM area data by **moto2ff.exe**.

Open the command prompt window and execute **moto2ff** in the command line as shown below.

```
C:\EPSON\gnu17>moto2ff StartAddress BlockSize InputFile
```

When the command is executed in the form shown above, *BlockSize* bytes of data are written out from *StartAddress* in *InputFile* to an output file (input file name + extension .saf). At this time, the blank addresses are filled with 0xff.

Example: To output 0x10000 bytes of data from the address 0x8000 in Motorola S3 format input.sa to a file input.saf

```
C:\EPSON\gnu17>moto2ff 8000 10000 input.sa
```

All data in the address range 0x8000 to 0x17fff in input.sa will be output to a file. Any blank address in this address range is filled with 0xff data.

### 3 SOFTWARE DEVELOPMENT PROCEDURES

Next, use **sconv32.exe** to convert the ROM area data file into the Motorola S2 format. For detailed information on **sconv32.exe**, refer to the sections in which other tools are described.

Open the command prompt window and execute **sconv32** in the command line as shown below.

```
C:\EPSON\gnu17>sconv32 S2 InputFile OutputFile
```

Example: To convert the Motorola S3 format file "input.saf" into the Motorola S2 format file "output.psa"

```
C:\EPSON\gnu17>sconv32 S2 input.saf output.psa
```

The file extension of the output file must be ".psa".

After creating the ROM data file according to the above procedure, be sure to perform the final verification of program operation on the actual target board using that file.

Select options using **winfog.exe** if the target model provides mask options. The **winfog** outputs the selected option information to a function option document file. For detailed information on **winfog.exe**, refer to the sections in which other tools are described.

Finally, pack the verified Motorola S2 ROM data file and the function option document file generated by **winfog** into a mask data file using **winmdc.exe**. The **winmdc** checks if the ROM area has been specified correctly in **moto2ff** in addition to packing files. For detailed information on **winmdc.exe**, refer to the sections in which other tools are described.

Present the mask data file generated by the above procedure to Seiko Epson.



## 3.4 Tutorial 2 (Using the User Makefiles)

In this section, as an example for using a user makefile, a user linker script file and a user command file, we'll take a look at a series of procedures, from building a project in the **IDE** to starting the debugger. For basic information on using the **IDE**, etc., refer to Tutorial 1.

### Files used

Tutorial 2 uses the sample files listed below that exist in the `c:\EPSON\gnu17\sample\S1C17common\simulator\simulatedIO` directory.

<code>vector.c</code>	Vector table file
<code>main.c</code>	C source file
<code>mymakefile.mak</code>	Makefile
<code>myldsfile.lds</code>	Linker script file
<code>mycmdfile.cmd</code>	Debugger command file (for simulator mode)
<code>myparfile.par</code>	Parameter file

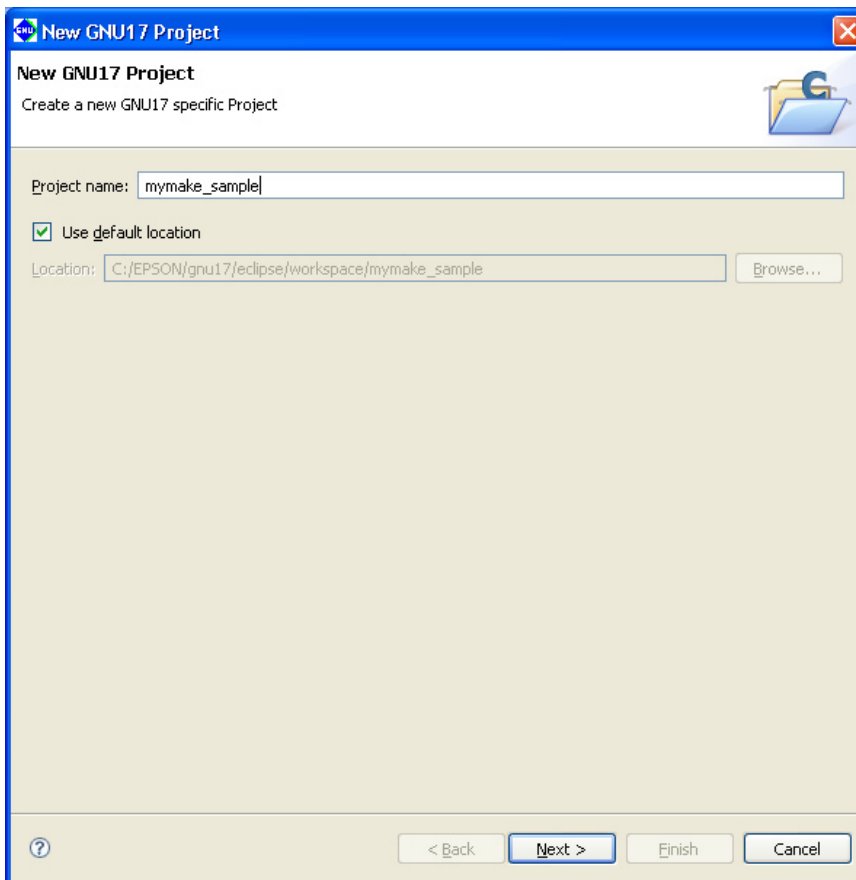
### 3.4.1 Creating a Project

First, create a new project with the **IDE**.

**Step 1:** Launch the **IDE**.

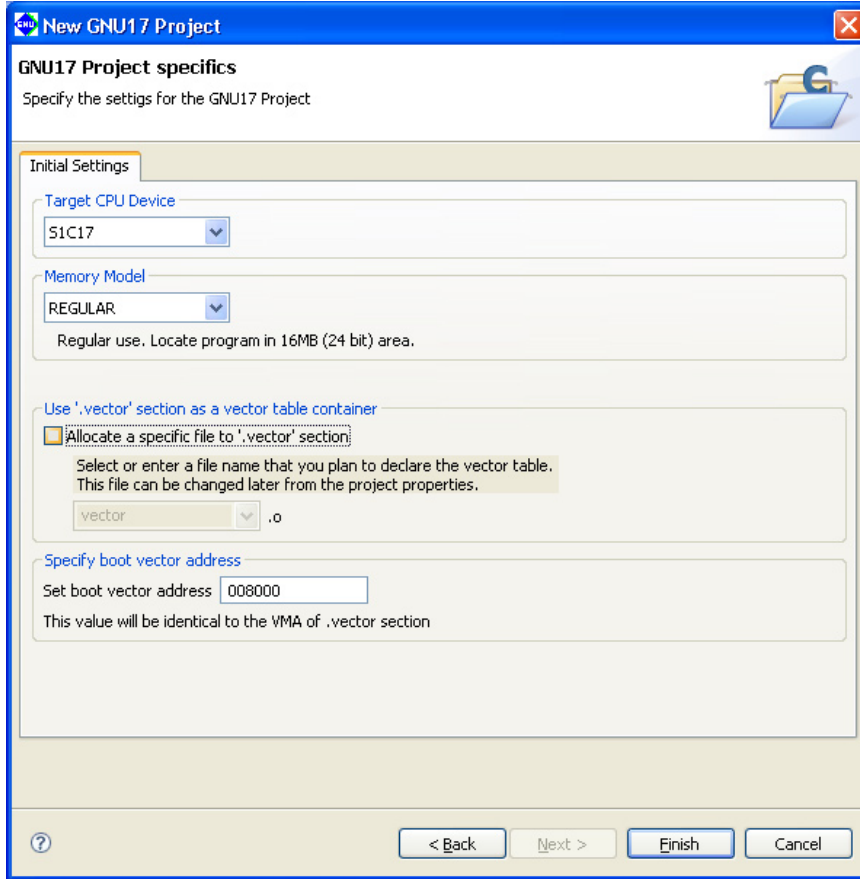
**Step 2:** Select [New] > [New GNU17 Project] from the [File] menu to start the [New GNU17 Project] wizard.

**Step 3:** Enter the project name "mymake\_sample" in the [Project name:] field.



In this tutorial, we create a project folder in the workspace (default). Leave the [Use default location] check box selected.

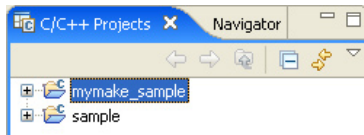
Step 4: Click the [Next>] button.



This tutorial does not use the makefiles and linker script files generated by the **IDE**, so there is no need to select the target CPU, memory model and `.vector` section.

Step 5: Deselect the check box [Allocate a specific file to `'.vector'` section].

Step 6: Click the [Finish] button to create a project.

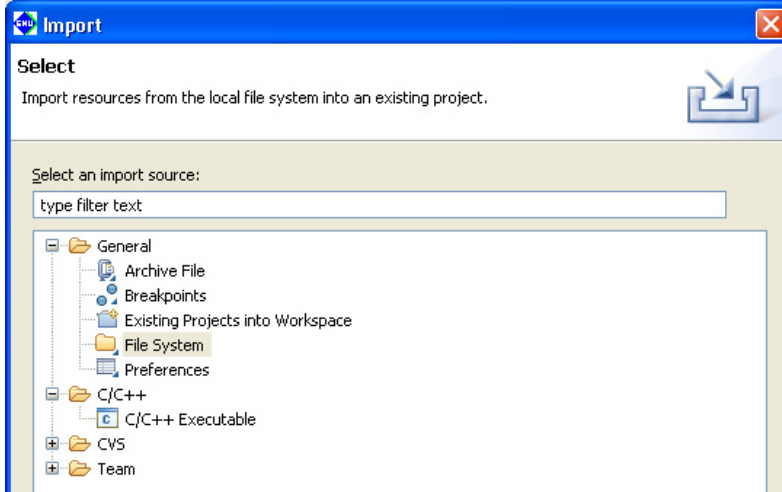


### 3.4.2 Importing Source Files

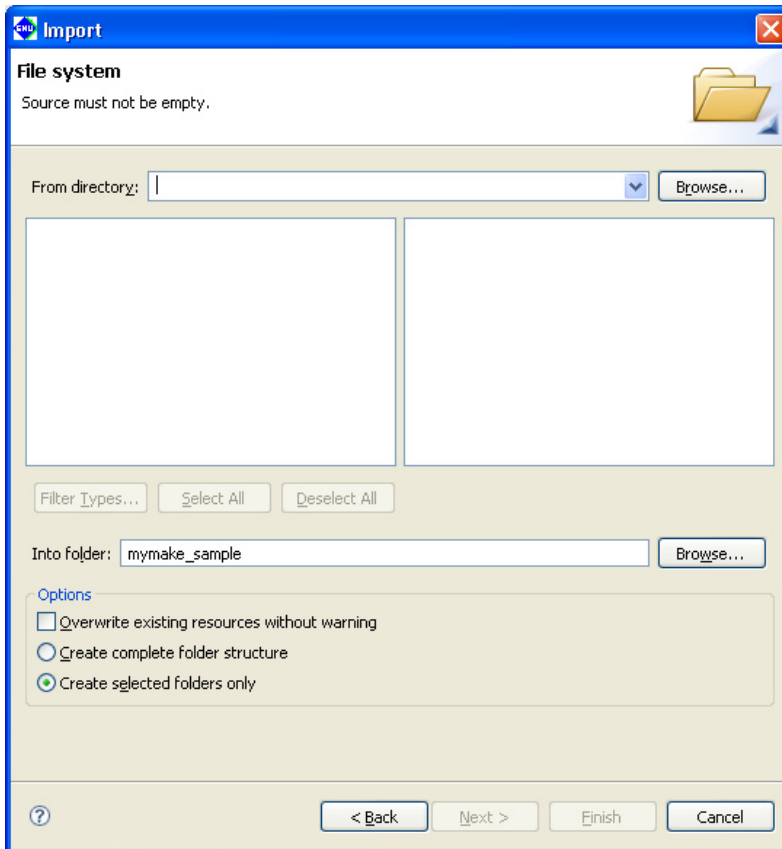
Import the source files after creating a project. Here, for the sake of convenience, we'll also import the makefiles stored in the same directory.

Step 7: Select "mymake\_sample" in [Navigator] view, and [Import...] from the [File] menu.

This launches the [Import] wizard.



Step 8: Select [General] > [File System] from the list displayed and click the [Next>] button.



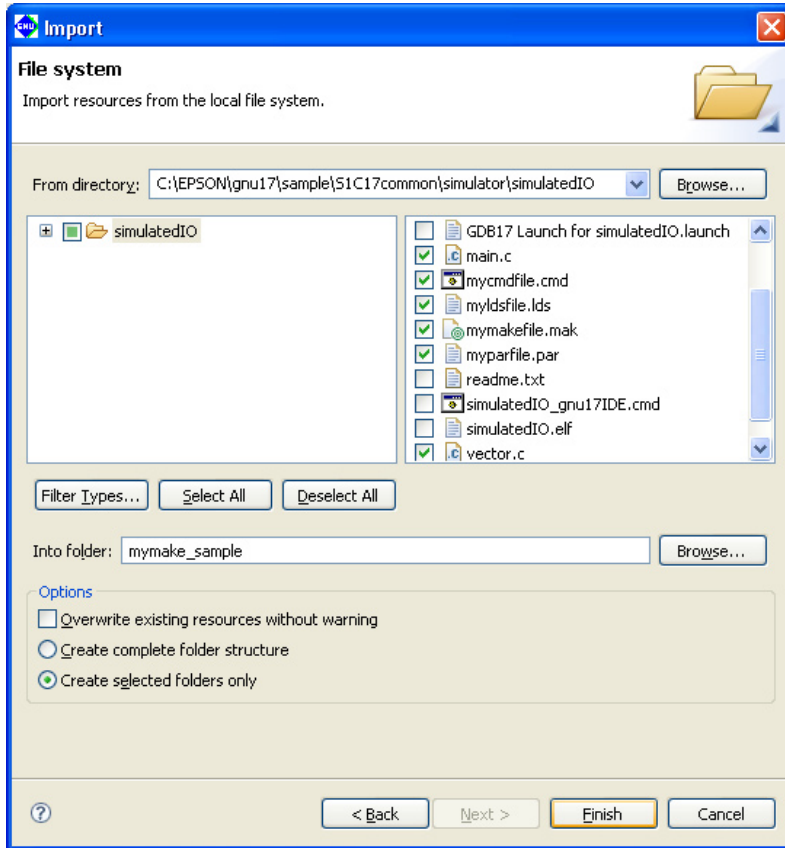
Step 9: Using the [Browse...] button in [From directory:], select the C:\EPSON\gnu17\sample\S1C17common\simulator\simulatedIO directory that contains the files to be imported.

### 3 SOFTWARE DEVELOPMENT PROCEDURES

The selected directory and the files in it will be displayed in the list boxes on the left and the right sides of the window, respectively.

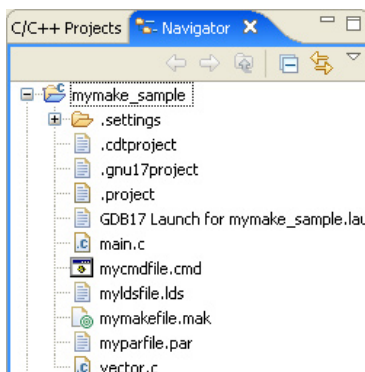
Step 10: Select the check boxes for the following files shown in the right-side list box.

<code>vector.c</code>	Vector table file
<code>main.c</code>	C source file
<code>mymakefile.mak</code>	Makefile
<code>myldsfiler.lds</code>	Linker script file
<code>mycmdfile.cmd</code>	Debugger command file
<code>myparfile.par</code>	Parameter file



Step 11: Click the [Finish] button.

You can inspect the files that have been added to the project from the [Navigator] view.



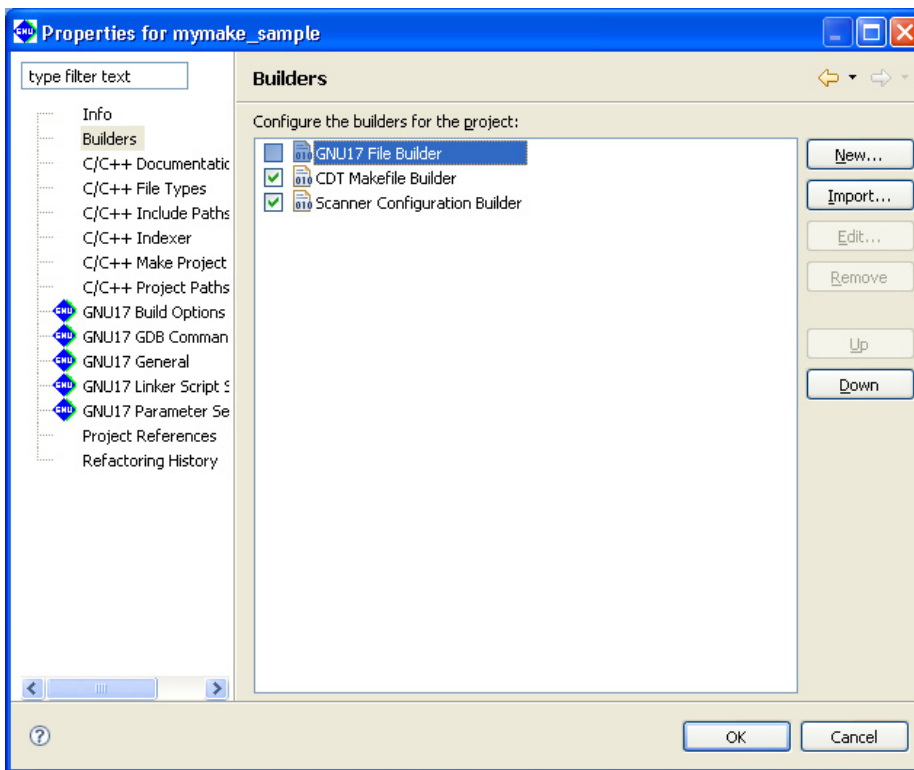
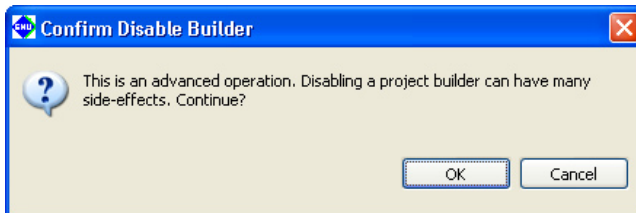
### 3.4.3 Disabling the GNU17 File Builder

The **IDE** is initialized to automatically generate makefiles, linker script files, parameter files, and debugger command files and to use these files when building a project or starting the debugger. In this tutorial, since we use separately prepared files, we need to change the default **IDE** settings to keep from using these files.

The following describes how to disable the file builder to prevent automatic generation of these files.

**Step 12:** After selecting the "mymake\_sample" project from the [Navigator] or the [C/C++ Projects] view, select [Properties] from the [Project] menu or context menu to display the [Properties] dialog box.

**Step 13:** Select [Builders] from the properties list and deselect the [GNU17 File Builder] check box. A dialog box below appears for confirmation. Click [OK].



**Step 14:** Click the [OK] button.

You can use your own makefiles, linker script files, parameter files, and debugger command files even without taking this step, but unnecessary files will be generated each time you build a project. Additionally, the automatically generated files will overwrite any current files with the same names.

Do not disable the file builder if any of the above files must be automatically generated by the **IDE**.

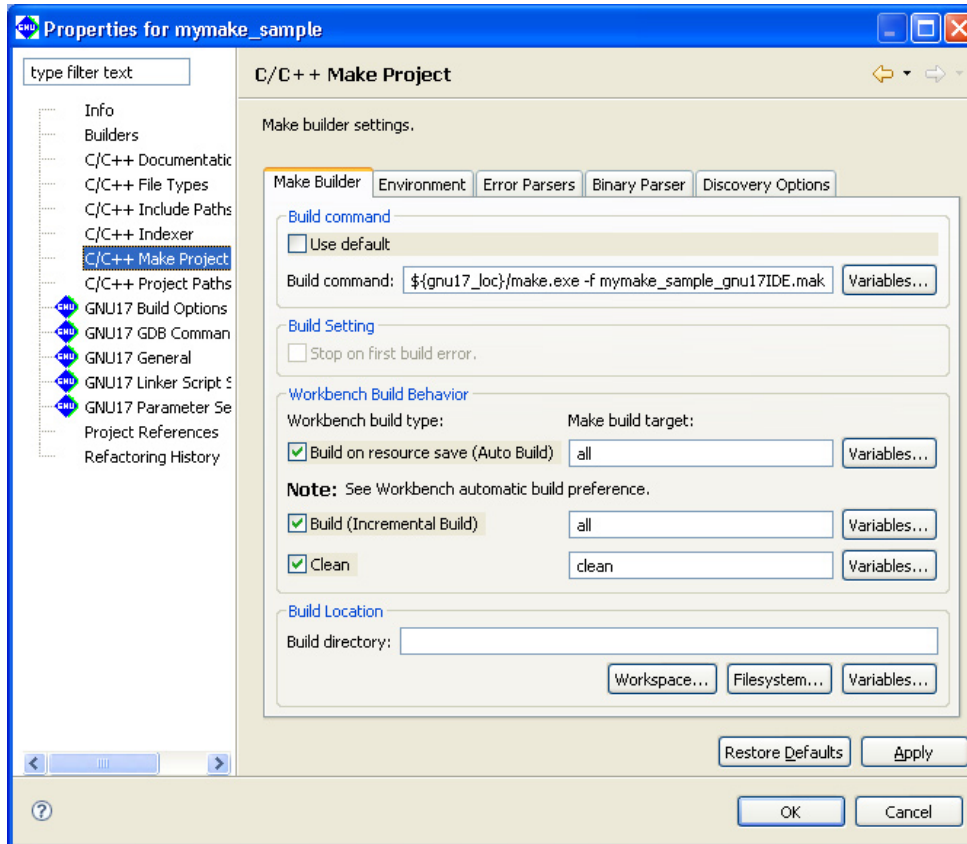
### 3.4.4 Setting and Correcting the Makefile

#### To specify a user makefile

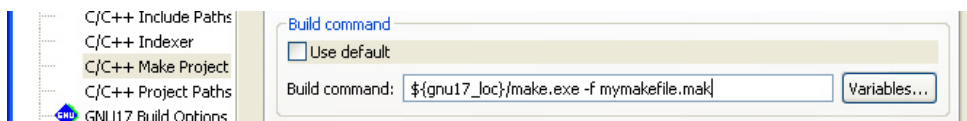
Now we'll set up the **IDE** to build the project using the separately prepared makefile. Here, we use `mymakefile.mak`, which we imported into the project.

Step 15: After selecting the "mymake\_sample" project from the [Navigator] or [C/C++ Projects] view, select [Properties] from the [Project] menu or context menu to display the [Properties] dialog box.

Step 16: Select [C/C++ Make Project] from the properties list to display the page for the [Make Builder] tab.



Step 17: Change the makefile name "mymake\_sample\_gnu17IDE.mak" set in [Build command:] to "mymakefile.mak".



No change is required if the makefile is `mymakefile.mak`. However, unless the target name in the user-created makefile is `all` (build) or `clean` (clean), the following settings must also be changed.

#### [Build (Incremental Build)]

Specify the target in the makefile to be called when executing a build process.

#### [Clean]

Specify the target in the makefile to be called when executing a clean process (to clear the generated files).

[Build on resource save (Auto Build)] is not used in the **IDE**.

Step 18: Click the [OK] button.

### Correcting the makefile contents

To use a makefile not created with the **IDE**, you may need to change the path written in it. If it contains a relative path, change it to a cygwin format path to allow file referencing from the project directory as well. You can use the **IDE** editor to make changes in the file.

Example: Before change: `SRCDIR= ..`

After change: `SRCDIR=/cygdrive/c/EPSON/gnu17/sample/S1C17common/simulator`

### 3.4.5 Building a Project

After setting the makefile, you can execute a build process as you would normally do. There is no need to set build options or to edit the linker script file.

Step 19: Select the project "mymake\_sample" name from the [Navigator] or the [C/C++ Projects] view.

Step 20: Select [Build Project] from the [Project] menu.

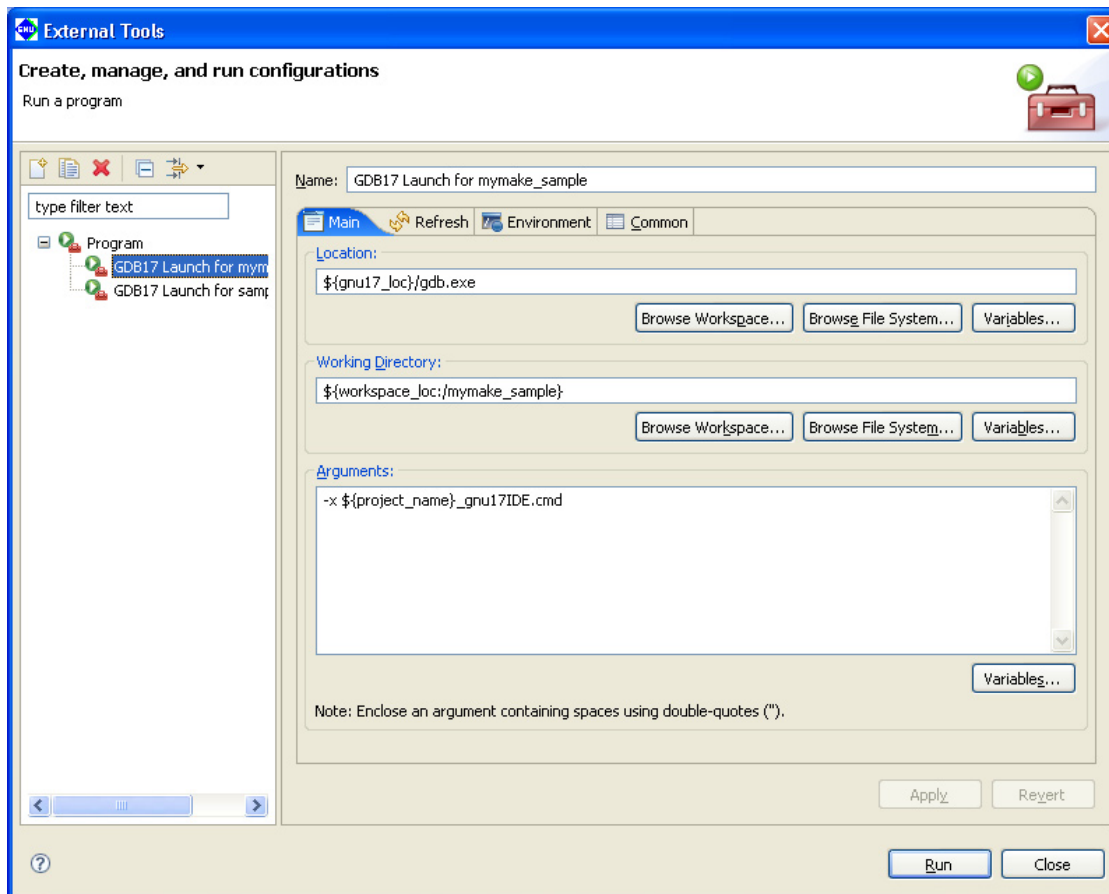
When the build command is selected this way, **make.exe** is executed with a specified makefile to generate the executable format object file "stdio.elf". (Since we are using a makefile that is not automatically generated, the object file is not named after the project name.)

### 3.4.6 Starting the Debugger

As the last step in Tutorial 2, described below is the method for making the necessary settings to execute the prepared command file at debugger startup.

Step 21: Select [External Tools] > [External Tools...] from the [Run] menu to display the [External Tools] dialog box.

Step 22: Select [GDB17 Launch for mymake\_sample] from the list in the dialog box and display the page for the [Main] tab.

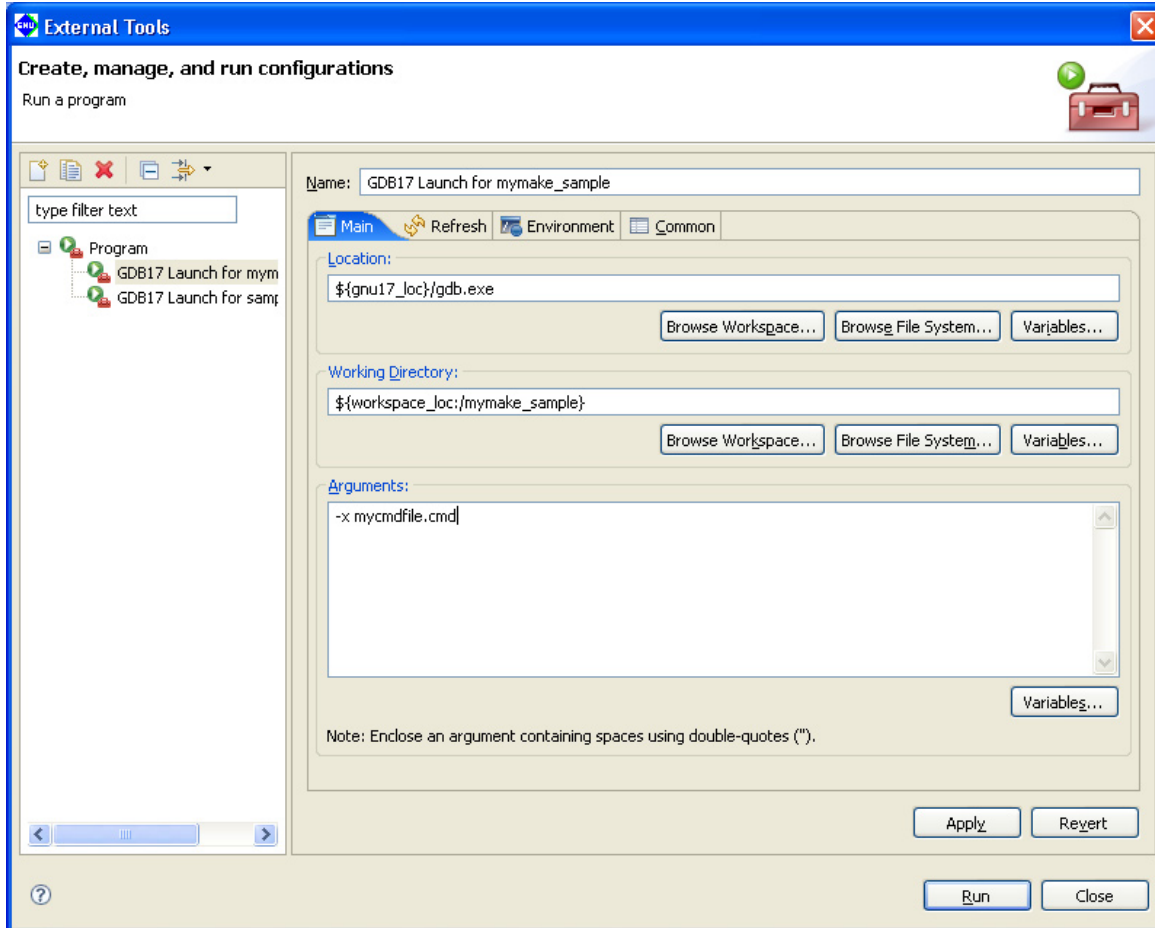


### 3 SOFTWARE DEVELOPMENT PROCEDURES

Step 23: In the [Arguments:] column, modify the `-x` option that specifies a command file as shown below.

Before change: `-x ${project_name}_gnu17IDE.cmd`

After change: `-x mycmdfile.cmd`



Step 24: Click the [Apply] button to confirm what you've altered here.

Starting the debugger will now execute `mycmdfile.cmd`.

Step 25: Click the [Close] button if you want to finish here.

To actually start the debugger, click the [Run] button.

See Tutorial 1 for basic debugger operations.



## 3.5 Tutorial 3 (Importing an IDE Project)

If you've already developed an application for the S1C17 Family with the **IDE**, you can continue with development or make revisions by importing that project into the **IDE** on another PC. Or you can develop another application, based on that project. The procedure for importing a project is explained here. For other procedures, refer to Tutorials 1 and 2.

Sample project directory used

```
C:\EPSON\gnu17\sample\S1C17common\simulator\simulatedIO
```

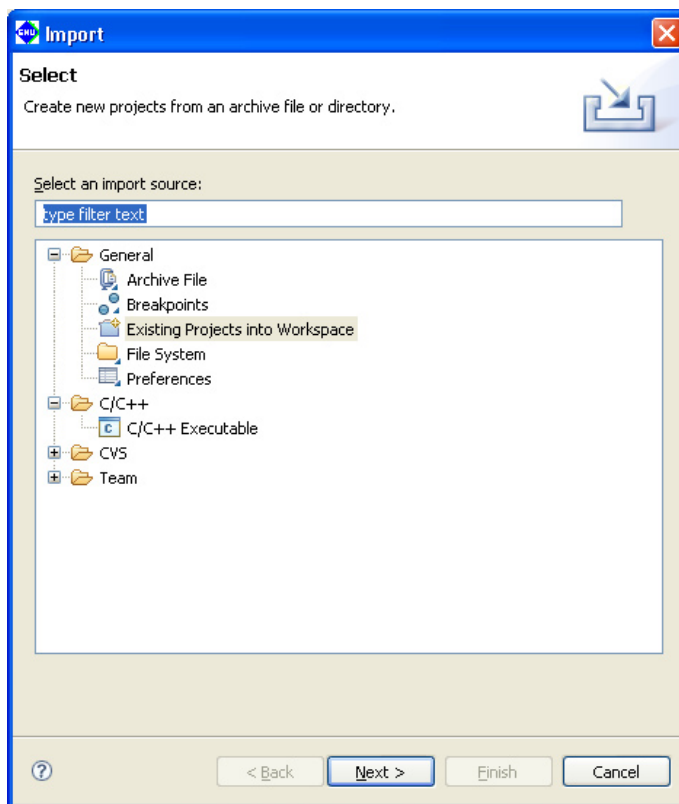
### To import a project

We'll assume that the project to be imported is copied to the HDD of your PC.

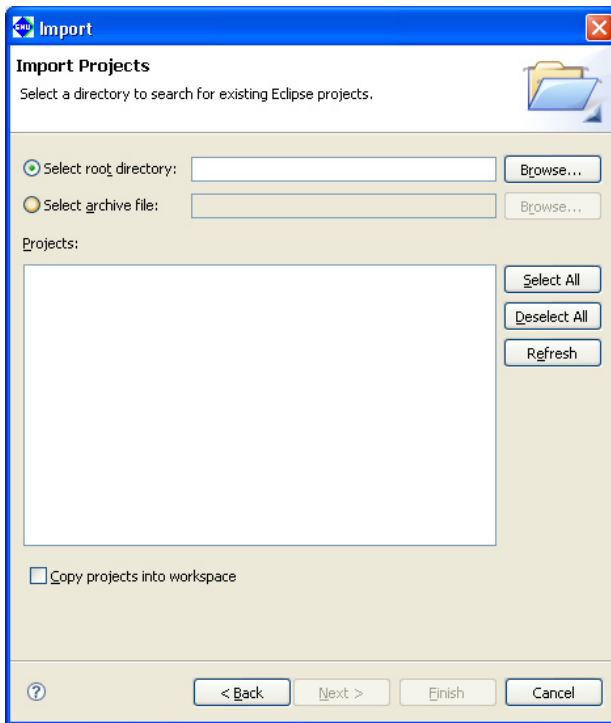
Step 1: Launch the **IDE**.

Step 2: Select [Import...] from the [File] menu.

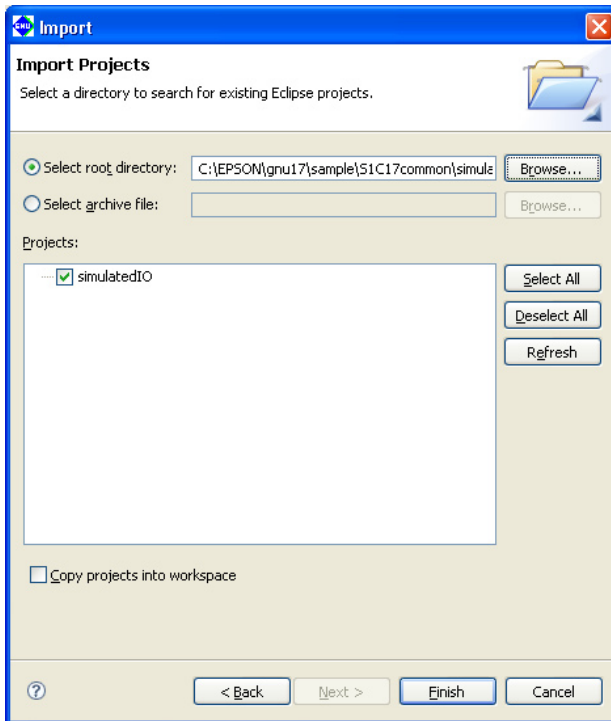
This launches the [Import] wizard.



Step 3: Select [Existing Projects into Workspace] from the displayed list and click the [Next>] button.



Step 4: Using the [Browse...] button in [Select root directory:], select the project directory C:\EPSON\gnu17\sample\S1C17common\simulator\simulatedIO to be imported.



Step 5: Select the [Copy projects into workspace] check box.

This will make a copy of the project into the workspace directory and the original project files will not be modified.

Step 6: Click the [Finish] button.

This imports the selected directory into the **IDE** as a project.

### About the directory structure and resource position

If all resources are stored together in a project directory, the project can be copied to any location without causing problems. The project can then be built at the copied destination with no further revisions.

Even if your project references certain external files or folders outside the project, you will not need to correct them as long as those files and folders are managed in the same directory structure. However, if makefiles, etc. are prepared externally and not the ones automatically generated by the **IDE**, as explained in Tutorial 2, the paths specified in these files may need to be corrected.

You will neither have a problem with the standard libraries and include directories as long as the tools are installed in the same directory where the original project was created in (e.g. C:\EPSON\gnu17). Otherwise, corrections are required for the user library and include directory. Make these corrections in the [GNU17 Build Options] of the [Properties] dialog box for the project.

## 3.6 Tutorial 4 (How to Use ES-Sim17)

The S5U1C17001C Package contains the Embedded System Simulator (**ES-Sim17**) for simulating the hardware functions of the target model, such as I/O ports and LCD display, during debugging on the PC. The debugger will launch **ES-Sim17** by setting some conditions with the **IDE**. This section describes operations on the **IDE** to use **ES-Sim17**. For other operations, see Tutorials 1 to 3. For details on **ES-Sim17**, see Section 10.10, "Embedded System Simulator (**ES-Sim17**)".

### 3.6.1 Settings Required for Launching ES-Sim17

To launch **ES-Sim17** at the start of the debugger, the following two settings are required.

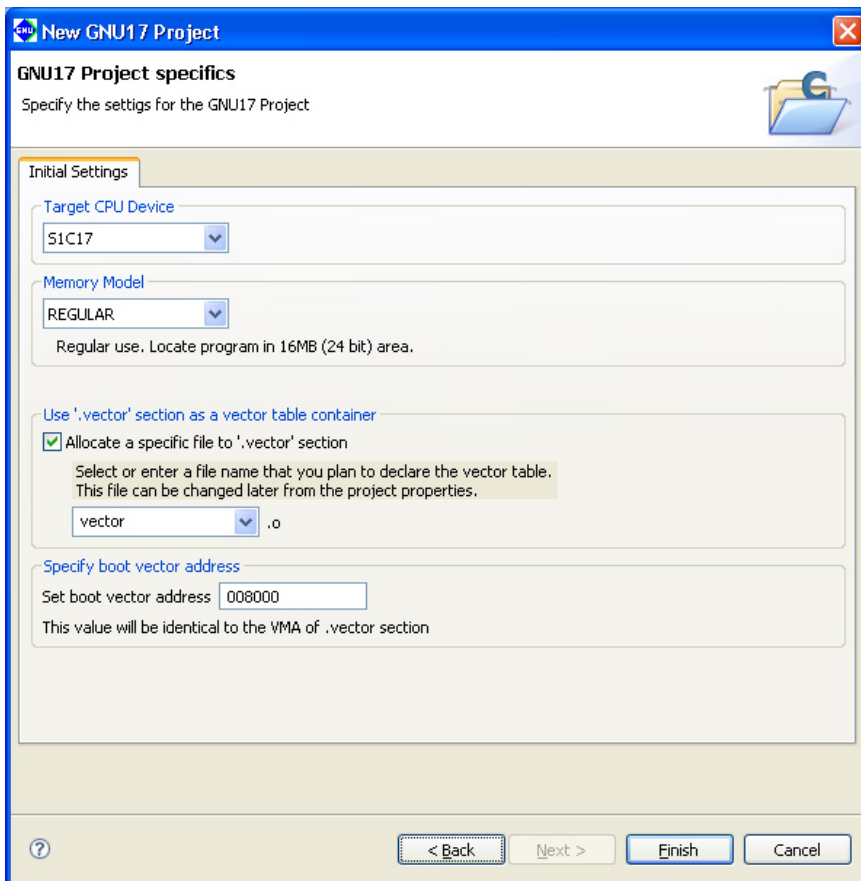
1. Specifying the target CPU (parameter file)
2. Starting the debugger in simulator mode (debugger command file)

This section explains the operations on the **IDE** to set the above conditions when a new project begins.

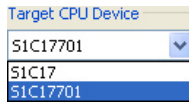
#### Specifying the target model

To simulate model specific hardware functions, the target CPU to be used must be specified in advance. When a new project is created, specify the target CPU as in the procedure below.

**Step 1:** Select [New GNU17 Project] from the [New] pull-down menu on the tool bar to launch the [New GNU17 Project] wizard. Then enter the project name in the [Project name:] box on the first wizard page and click the [Next>] button.



Step 2: Select "S1C17701" from the [Target CPU Device] combo box.



"S1C17" is provided for simulating the S1C17 Core only and selecting it will not launch **ES-Sim17** at the start of debugging.

Select "S1C17" when debugging the program in simulator mode without **ES-Sim17**. When debugging the program using an ICD (in ICD Mini mode), **ES-Sim17** will not start up regardless of how the target CPU has been selected.

The models displayed in the list may be added/deleted by the configuration file that will be modified when a new model is released or an existing model is discontinued.

Step 3: Configure the memory model and vector section as necessary, then terminate the wizard by clicking the [Finish] button.

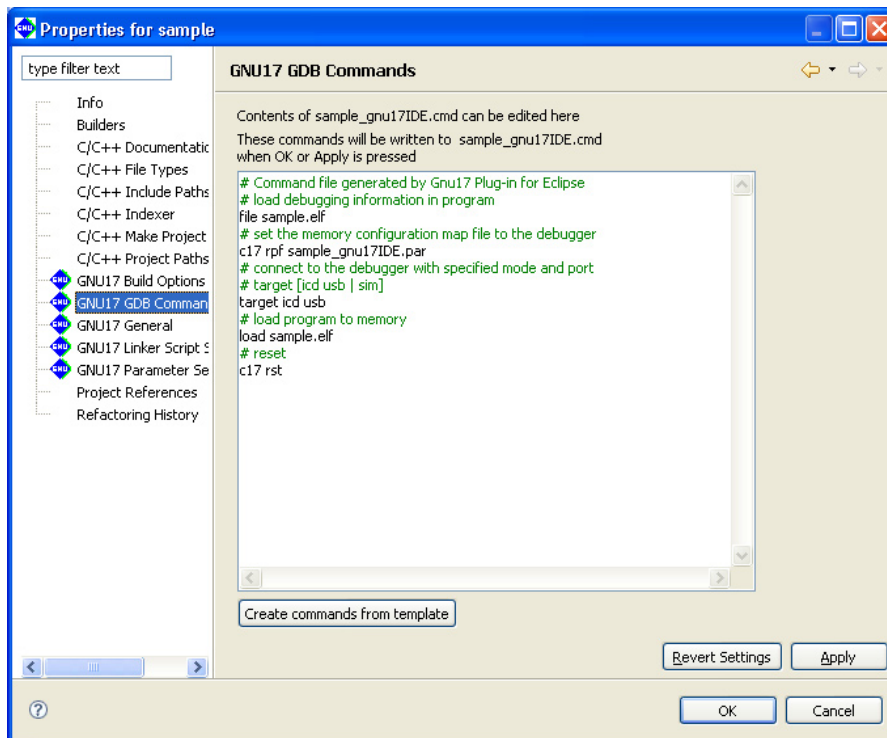
After that, edit source files and build them as usual.

The target CPU can also be specified on the [GNU17 General] page of the [Preferences] dialog box similar to above.

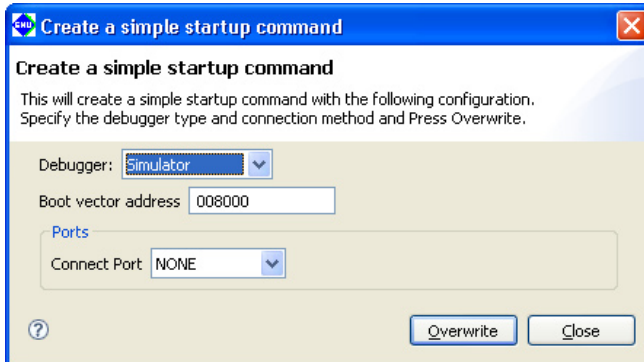
### Creating a debugger command file for simulator mode

The **ES-Sim17** will be able to run only when the debugger is running in simulator mode. The following shows the operations on the **IDE** to launch the debugger in simulator mode.

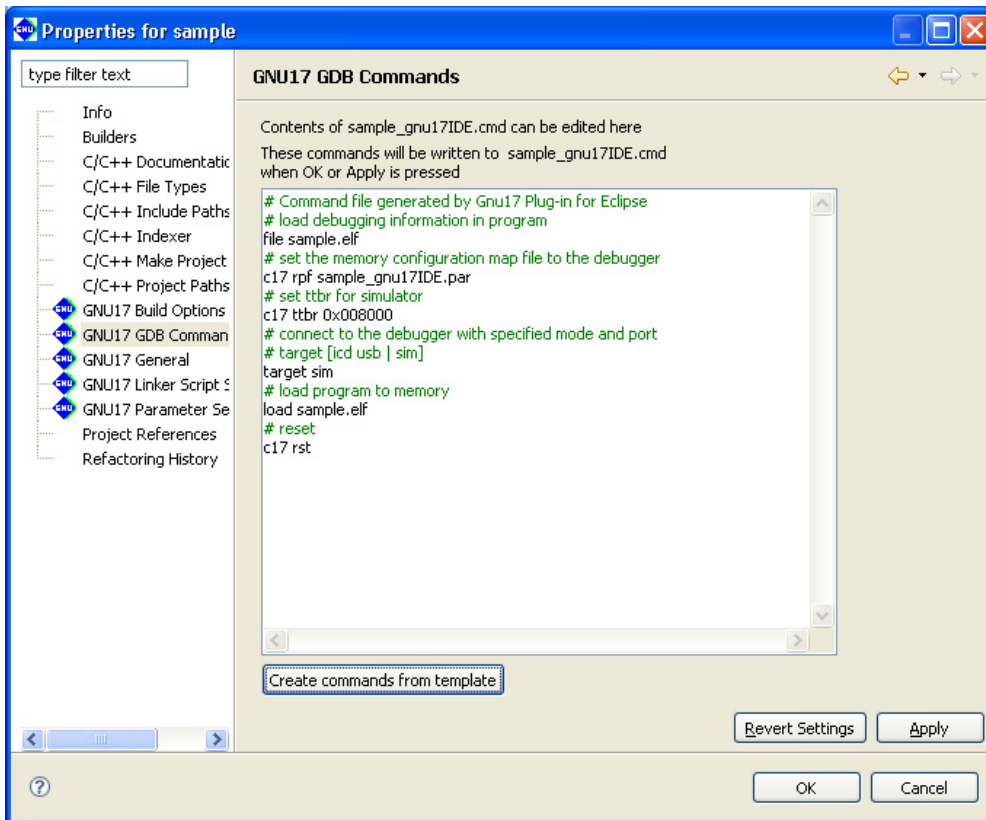
Step 4: Display the [Properties] dialog box by selecting [Properties] from the [Project] menu and open the [GNU17 GDB Commands] page.



Step 5: Display the [Create a simple startup command] dialog box by clicking the [Create commands from template] button and select "Simulator" from the [Debugger:] combo box.



Step 6: Click the [Overwrite] button.



Step 7: Click the [OK] button.

Following the above steps the settings for launching **ES-Sim17** at the same time the debugger starts up are finished. Other operations are not necessary in the source edit and build stages to launch **ES-Sim17**.

### 3.6.2 How to Launch ES-Sim17 in the Existing Project

This section explains the procedure from importing the project, which has been made for the S1C17 Core as the target, to launching the debugger and **ES-Sim17** after changing the target model.

Sample project directory used

```
C:\EPSON\gnu17\sample\S1C17701\simulator\application
```

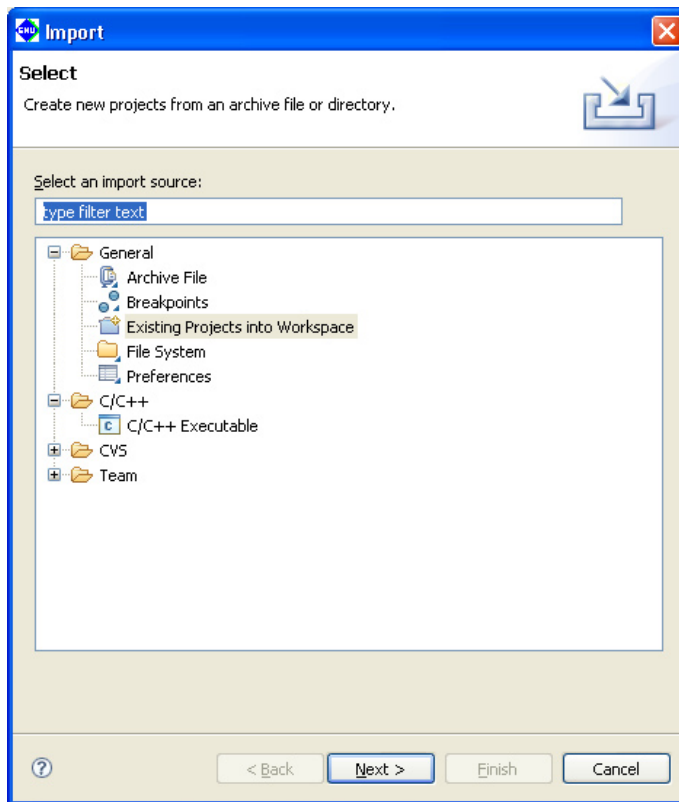
#### Importing the project

This tutorial assumes that the project to be imported is copied to the HDD of your PC.

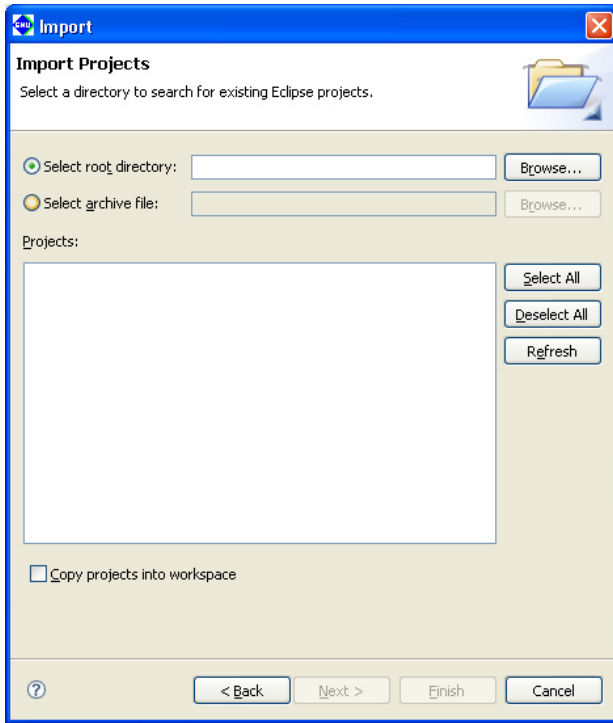
Step 1: Launch the **IDE**.

Step 2: Select [Import...] from the [File] menu.

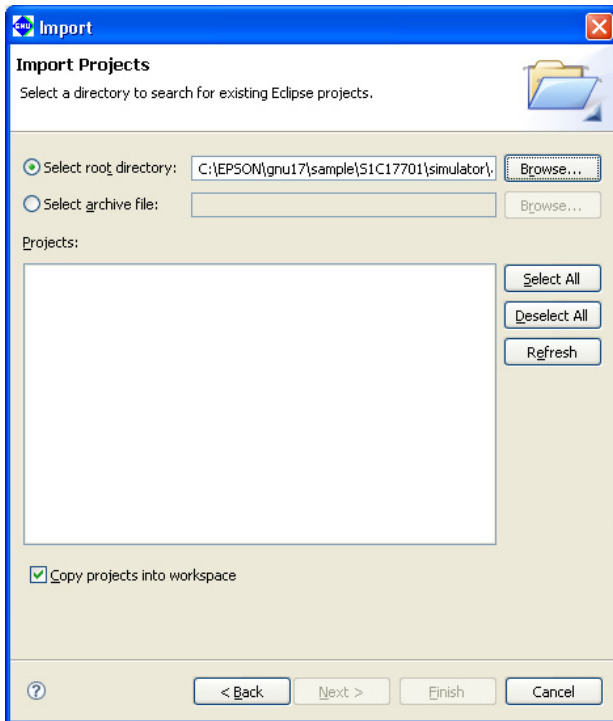
This launches the [Import] wizard.



Step 3: Select [Existing Projects into Workspace] from the displayed list and click the [Next>] button.



Step 4: Using the [Browse...] button in [Select root directory:], select the project directory C:\EPSON\gnu17\sample\S1C17701\simulator\application to be imported.



Step 5: Select the [Copy projects into workspace] check box.

This will make a copy of the project into the workspace directory and the original project files will not be modified.

Step 6: Click the [Finish] button.

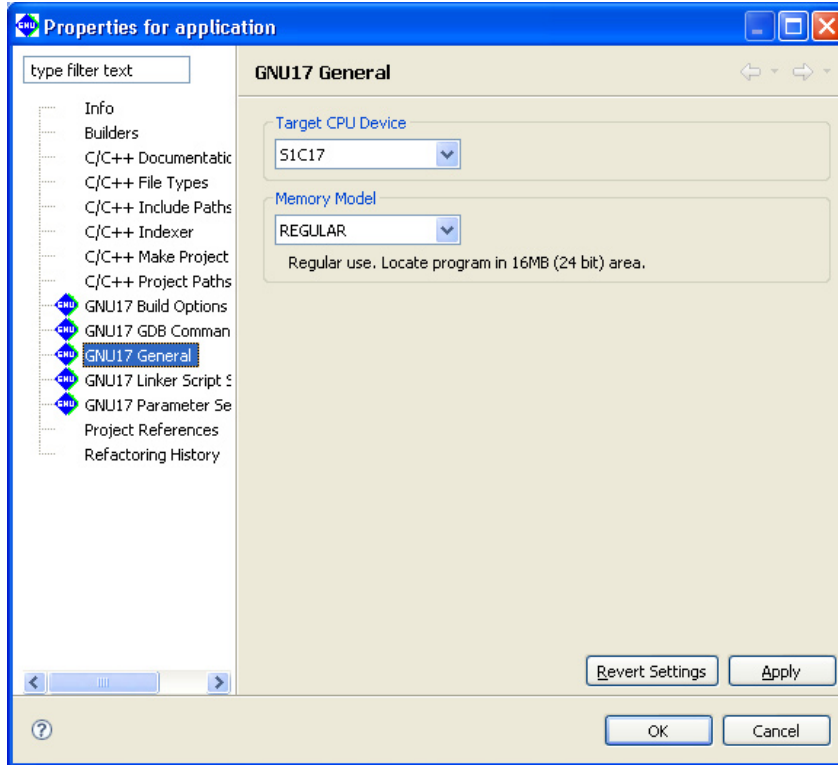
This imports the selected directory into the **IDE** as a project.



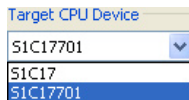
## Changing the target CPU (parameter file)

The sample project has been made for the S1C17 Core as the target CPU. This must be changed to the target processor to be simulated by **ES-Sim17**.

Step 7: Display the [Properties] dialog box by selecting [Properties] from the [Project] menu and open the [GNU17 General] page.



Step 8: Select "S1C17701" from the [Target CPU Device] combo box.



Step 9: Click the [OK] button.

Following the above steps the target model name for launching **ES-Sim17** with the debugger is written to the parameter file that will be passed to the debugger as shown below.

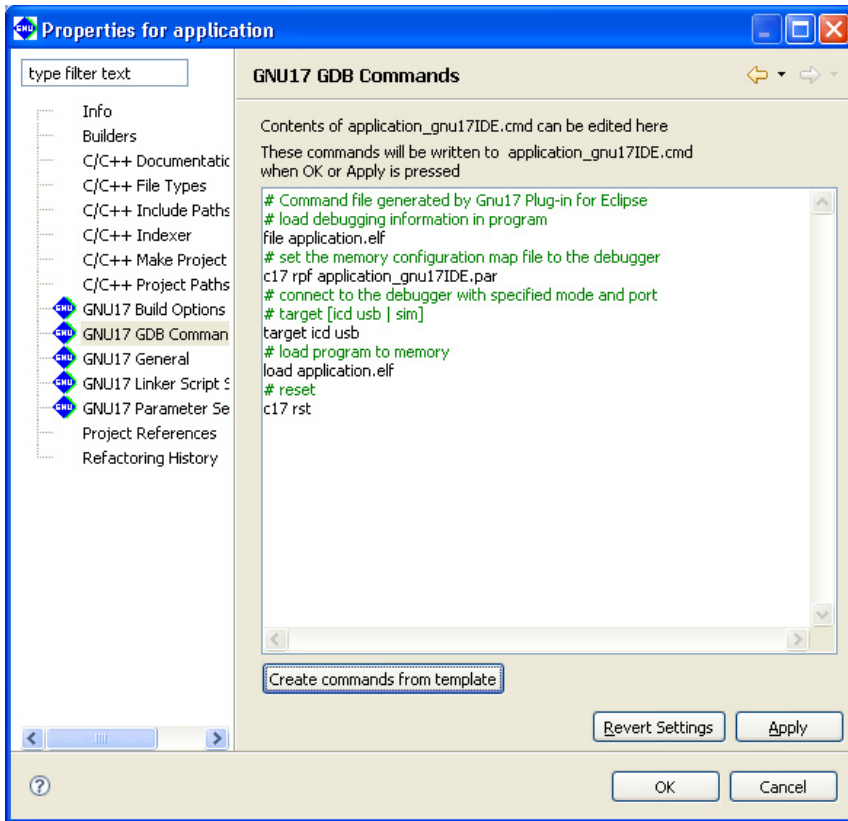
```
ESSIM S1C17701 ("S1C17701" is the selected target model name.)
```

\* When the user provides the original parameter file, add the above description to the file.

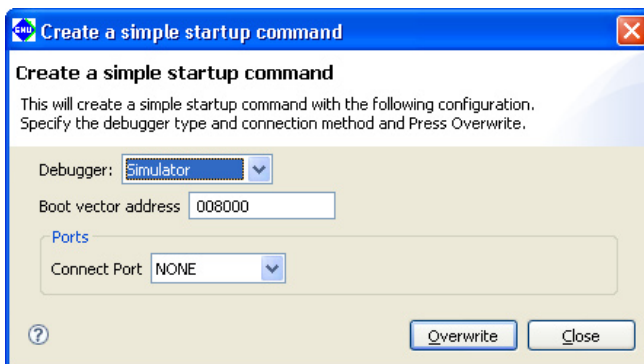
### Edit the debugger command file to one for simulator mode

The ES-Sim17 starts up when the debugger enters simulator mode. The command file in the sample project is made to set the debugger to ICD Mini mode, therefore, it must be changed to simulator mode.

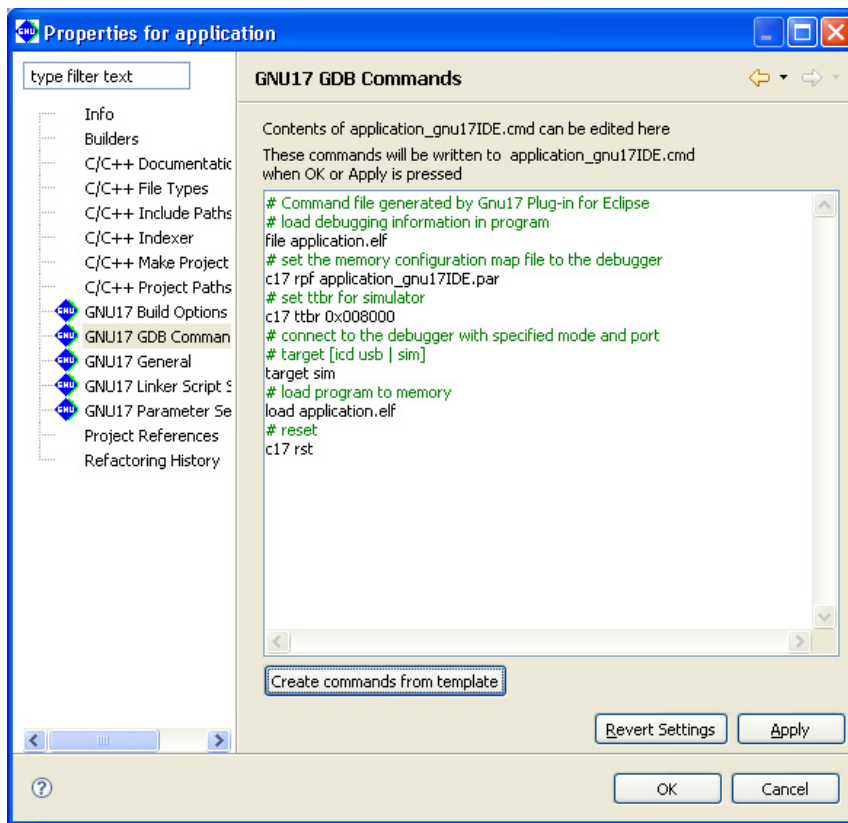
Step 10: Display the [Properties] dialog box by selecting [Properties] from the [Project] menu and open the [GNU17 GDB Commands] page.



Step 11: Display the [Create a simple startup command] dialog box by clicking the [Create commands from template] button and select "Simulator" from the [Debugger:] combo box.



Step 12: Click the [Overwrite] button.



Step 13: Click the [OK] button.

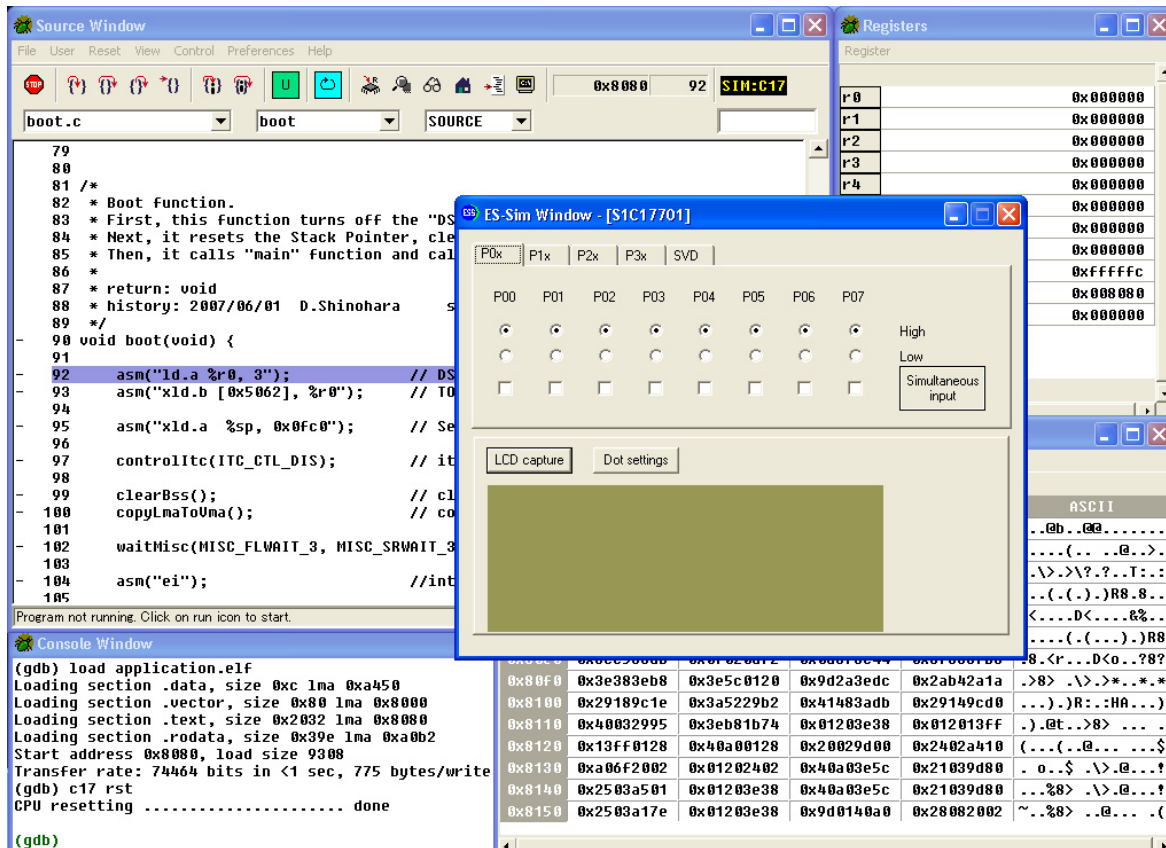
Following the above steps the command for setting the debugger to simulator mode (for launching **ES-Sim17**) is written to the command file that will be passed to the debugger as shown below.

```
target sim
```

- \* When the user provides the original command file that contains the "target icd usb" command, rewrite the command to "target sim".

## Launching the debugger

Step 14: Select [External Tools] > [GDB17 Launch for application] from the [Run] menu.



The debugger starts up and enters simulator mode. At the same time the [ES-Sim] window opens.

For the functions and how to operate **ES-Sim17**, see Section 10.10, "Embedded System Simulator (**ES-Sim17**)".

## 3.7 Debugging Environment

The debugger supports two connect modes, of which the mode used is set by the **target** command.

### ICD Mini mode

In this mode, the ICD Mini (S5U1C17001H) or ICD board is used to perform debugging. The program is executed on the target board.

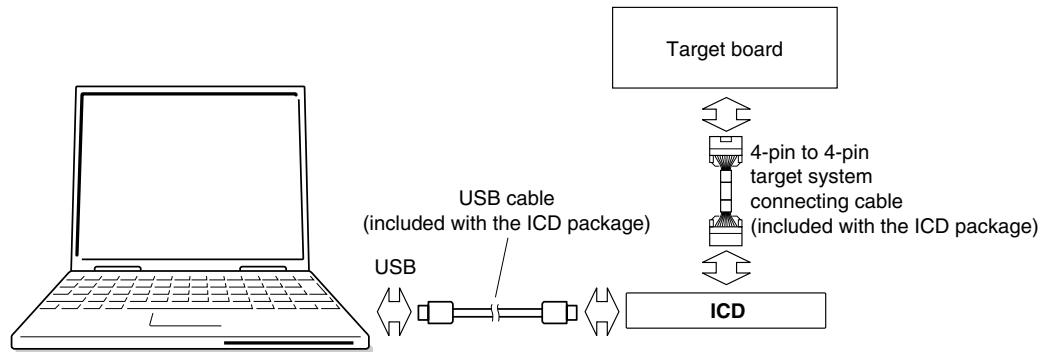


Figure 3.7.1 Example of debugging system using an ICD

#### Specification method

Command: (gdb) **target icd usb**

Specification in **IDE**:

Select "ICD Mini" from the [Debugger:] combo box in the [Create a simple startup command] dialog box to generate a startup command file.

To start in ICD Mini mode, make sure an ICD and target board are connected correctly, and that the power for these units is turned on. For details on how to use the ICD, refer to the manual for the ICD used.

Note that the trace function is not available in ICD Mini mode.

### Simulator (SIM) mode

In simulator mode, target program execution is simulated in internal memory of a personal computer, with no other tools required. However, the ICD-dependent functions cannot be used in this mode.

#### Specification method

Command: (gdb) **target sim**

Specification in **IDE**:

Select "Simulator" from the [Debugger:] combo box in the [Create a simple startup command] dialog box to generate a startup command file.

The trace function is available in simulator mode. The flash writer function cannot be used.

## 3.8 Sections and Linkage

---

Here, the concept of section management that is required when you create and link source files is explained.

The source file contains data with various attributes, such as program code, constants, and variables. In an embedded system, data management must assume that data will be mapped to different memory devices such as ROM and RAM. For this reason, logical areas called "sections" are provided to enable management of data with their attributes.

For example, if a program is created on the assumption that program code present in multiple source files will be located in one section, program code can easily be combined from these source files when linked, and will consequently be located in the same ROM. And since addresses can be specified separately for each file, they can be located on separate devices, such as internal ROM and external ROM.

Four broad categories (attributes) of sections are set in the **xgcc** C compiler, and data is located in the appropriate sections according to the contents of the source files.

(1) **.text** section

Program code is located here. All code is eventually written to ROM.

(2) **.data** section

Read/writable data with initial values are located here. The data is written to ROM, from which it is transferred to RAM before use.

(3) **.rodata** section

Variables defined with `const` are located here. They are eventually written to ROM.

(4) **.bss** section

Variables without initial values are located here. Memory is allocated without a specific value.

**.vector** section

The **IDE** has another section with `.rodata` attribute, the `.vector` section, available for use for vector tables.

For C sources, create a vector table with a `const` declaration and locate its object in the `.vector` section.

For the assembler sources, a vector table may be written in `.rodata` or the `.text` section. However, if a vector table is located in the `.text` section, you must change the `.vector` section attribute to `.text`.

For more information, refer to Section 5.7.6, "Editing a Linker Script".

Discussed below is the relationship between sections and actual memory locations.

#### Example source files

##### (file1.s)

---

```
.section .rodata                ; .rodata section

.global BOOT
.align 2
.long BOOT                      ; 0x00 reset
.long UNALIGN                   ; 0x01 unalign
.long EXCEPTION                 ; 0x02 nmi
    :
    :

.text                          ; .text section
BOOT:
    xld.a %sp, 0x3f00          ; set SP
    :
    :
```

##### (file2.s)

---

```
    :
    .data                      ;.data section
    :
    :
    .rodata                    ;.rodata section
    :
    :
```

##### (file3.c)

---

```
    :
#include <string.h>

int    i_bss;                  /* .bss section */
int    i_data = 1;             /* .data section */
const  int  i_rodata = 0x12345678; /* .rodata section */
const  char sz_rodata[] = "ABCDEFGH"; /* .rodata section */
    :

int main()                      /* .text section */
{
    char sz_buf[10];
    int  i;

    for( i = 0; i < 5; ++i ){
        sz_buf[i] = sz_rodata[i];
    }

    :
    :
    return 0;
}
```

## Example linker script file

```

(sample.lds)
/* Linker Script file */

SECTIONS
{
  /* location counter */
  . = 0x0;                                     ... (1)

  /* section information */
  .bss 0x000000 :                               ... (2)
  {
    __START_bss = . ;                          ... (3)
    file1.o(.bss)                               ... (4)
    file2.o(.bss)
    file3.o(.bss)
    C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
    C:/EPSON/gnu17/lib/24bit/libgcc.a(.bss)
    C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
    __END_bss = . ;                             ... (5)
  }

  .data __END_bss : AT( __END_rodata )         ... (6)
  {
    __START_data = . ;
    file1.o(.data)
    file2.o(.data)
    file3.o(.data)
    C:/EPSON/gnu17/lib/24bit/libc.a(.data)
    C:/EPSON/gnu17/lib/24bit/libgcc.a(.data)
    C:/EPSON/gnu17/lib/24bit/libc.a(.data)
    __END_data = . ;
  }

  .vector 0x008000 :                            ... (7)
  {
    __START_vector = . ;
    file1.o(.rodata)
    __END_vector = . ;
  }

  .text __END_vector :                          ... (8)
  {
    __START_text = . ;
    file1.o(.text)
    file2.o(.text)
    file3.o(.text)
    C:/EPSON/gnu17/lib/24bit/libc.a(.text)
    C:/EPSON/gnu17/lib/24bit/libgcc.a(.text)
    C:/EPSON/gnu17/lib/24bit/libc.a(.text)
    __END_text = . ;
  }

  .rodata __END_text :                          ... (9)
  {
    __START_rodata = . ;
    file2.o(.rodata)
    file3.o(.rodata)
    C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
    C:/EPSON/gnu17/lib/24bit/libgcc.a(.rodata)
    C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
    __END_rodata = . ;
  }

  /* load address symbols */
  __START_data_lma = LOADADDR( .data );        ... (10)
  __END_data_lma = __START_data_lma + SIZEOF( .data );
}

```



The example source files shown above include the following sections.

```
file1    .rodata and .text sections
file2    .data and .rodata sections
file3    .text, .bss, .data, and .rodata sections
```

These sections are relocated according to the `SECTIONS` command specified in a linker script file. The contents of the example linker script file are described below.

- (1) Set the location counter to 0x0. This location is assumed to be address 0x0. The location counter will be incremented by specifying an address or locating sections. `'.'` is used to reference the current location counter value in a linker script.
- (2) Define the `.bss` output section to be output to an executable format object file. The `.bss` section begins with address 0x000000 as the definition specifies the address.
- (3) The linker script created by the **IDE** contains symbols defined to indicate a section start address. In this example, the `.bss` output section start address is defined as the symbol name `__START_bss`. `'.'` represents the location counter value, so the symbol is defined with the value 0x000000. Other sections have a start address definition similar to this. These symbols can be referenced from program source files as global symbols.
- (4) Specify the object files with their basic section attribute to be located in this section. In this example, the `.bss` sections in the specified files will be located in order of `file1.o`, `file2.o`, and `file3.o` in the `.bss` output section. The `file2.o` does not contain a `.bss` section, so no memory area will be allocated but no error will occur even if it is specified like this.

Furthermore, library files must be written if the application uses library functions. In this example, `libgcc.a` cannot reference symbols in the preceding `libc.a`, therefore `libc.a` is specified twice to resolve references to unknown symbols (some functions in `libgcc.a` call a function in `libc.a`). This specification does not locate the actual code twice.

- (5) The symbol `__END_bss` is defined to indicate the `.bss` output section end address similar to (3) above. The location counter value specified by `'.'` is `__START_bss + (total size of all .bss sections located)`.
- (6) Define the `.data` output section. With `__END_bss` specified as the start address, this section (VMA) is located immediately after the `.bss` section. All `.data` sections in the input files are placed into this output section.  
The defined section is located at the VMA (memory address accessed when actually executing code or when reading/writing data). The VMA is normally the same as the LMA (load memory address in which data is stored). However, the `.data` section requires that initial values be written to ROM and that the initial values be copied to RAM before use. For this reason, the LMA (ROM address) must be specified separately. The `AT` statement specifies that the actual code in the `.data` section must be located from `__END_rodata` (immediately following the `.rodata` section).
- (7) Define the `.vector` output section beginning with address 0x008000. In this example, the vector table is written in the `.rodata` section defined in `file1.o`, so only `file1.o(.rodata)` is specified as the file (section attribute) to be located in this section.
- (8) Define the `.text` output section immediately following the `.vector` section. All `.text` sections in the input files are placed into this output section.
- (9) Define the `.rodata` output section immediately following the `.text` section. All `.rodata` sections in the input files are placed into this output section. The `file1.o` is not specified here, as its `.rodata` section has been located in the `.vector` section.
- (10) The `__START_data_lma` and `__END_data_lma` are defined for the start (`LOADADDR(.data)`) and the end addresses (LMA) of the `.data` section in which the actual data is stored. These symbols are used to copy data from the LMA to the VMA in the program source.

Figure 3.8.1 shows the memory map configured by this example script.

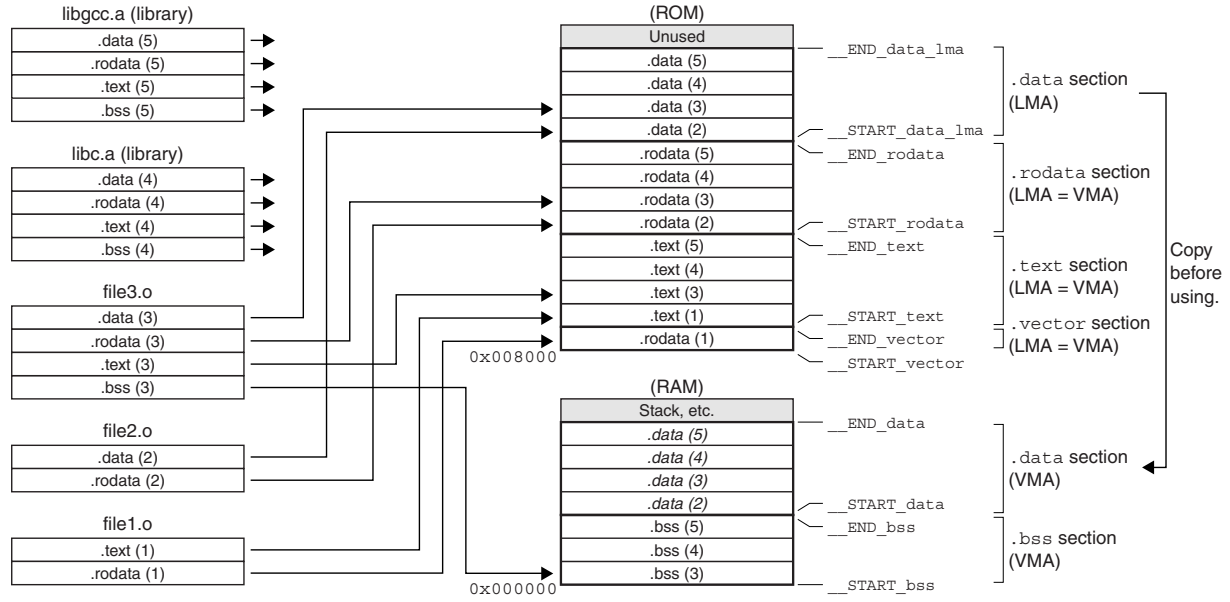


Figure 3.8.1 Memory map configured by sample.lds

# 4 Source Files

This chapter explains the rules and grammar involved with the creation of source files.

## 4.1 File Format and File Name

---

Use the **GNU17 IDE** editor or a general-purpose editor to create source files.

### File format

Save data in a standard text file.

### File name

C source file            <*file name*>.c

Assembly source file <*file name*>.s

Specify the <*file name*> with not more than 32 alphanumeric characters shown as follows:

a-z, A-Z, 0-9 and \_

This rule applies to file names for all the S1C17 tools.

### Directory name

Only alphanumeric characters can be used for directory names just as for file names. Do not use spaces or other symbols. Up to 64 characters can be used for a path name including directory and file names.

### Global variables/static variables

Up to 200 characters can be used to name global and static variables.

A total of 32,000 global and static variables can be accepted.

### File size

The following shows the guide about the upper limit of the C source file size:

- In the case of a source file that contains only variables, constants and arrays, up to 100,000 lines can be accepted.
- In the case of a source file that contains only executable codes (not including arrays and variables), up to 20,000 lines can be accepted. However, the number of acceptable lines varies depending on the source density.
- Consider these two conditions above as reference for sources in which variables, constants, arrays and executable codes are mixed.
- The number of lines shown above varies depending on compile environment conditions. Moreover, the compiler may be forcibly terminated due to insufficient resources. In this case, build the program under a resource-rich environment or divide the source file into multiple files before compiling. (Resources described here depend on the OS used rather than the RAM capacity of the PC.)
- Up to 512 characters can be used per line in C source files.

In the case of assembly source files, up to 30,000 lines can be accepted.

### Tab setting

The recommended tab stop is every 4 characters. This is the default tab setting when the debugger displays sources. Source display/mixed display with the **gdb** debugger of a source set at a tab interval other than of 4 characters may result in a displaced output of the source part. In this case, change the tab setting of the debugger according to that of the source using the [Source Preference] dialog box that be selected from the [Preference] menu (can be specified from 1 to 16 characters).

### EOF

Make sure that each statement starts on a new line and that EOF is entered after line feed (so that EOF will stand independent at the file end).

## 4.2 Grammar of C Source

The **xgcc** C compiler included in this package is the GNU C Compiler (ver. 3.3.2) under ANSI C standards. Make sure C sources are created according to ANSI C standards. If you want information about the syntax, please refer to ANSI C textbooks generally available on the market.

### 4.2.1 Data Type

The **xgcc** C compiler supports all data types under ANSI C. The size of each data type (in bytes) and the effective range of values that can be expressed are listed in Table 4.2.1.1.

Table 4.2.1.1 Data type and size

Data type	Size	Effective range of a number
char	1	-128 to 127
unsigned char	1	0 to 255
short	2	-32768 to 32767
unsigned short	2	0 to 65535
int	2	-32768 to 32767
unsigned int	2	0 to 65535
long	4	-2147483648 to 2147483647
unsigned long	4	0 to 4294967295
long long int	8	-9223372036854775808 to 9223372036854775807
unsigned long long int	8	0 to 18446744073709551615
pointer	4	0 to 16777215
float	4	1.175e-38 to 3.403e+38 (normalized number)
double	8	2.225e-308 to 1.798e+308 (normalized number)
long double	8	2.225e-308 to 1.798e+308 (normalized number)
long long	8	-9223372036854775808 to 9223372036854775807
unsigned long long	8	0 to 18446744073709551615
wchar_t	2	0 to 65535

The `float` and `double` types conform to the IEEE standard format.

Handling of `long long`-type constants requires the suffix `LL` or `ll` (`long long` type) or `ULL` or `ull` (`unsigned long long` type). If this suffix is not present, an error is assumed, since the compiler may not be able to recognize `long long`-type constants as such.

```
Example: long long ll_val;
         ll_val = 0x1234567812345678;
           →error: integer constant is too large for "long" type
         ll_val = 0x1234567812345678LL;
           →OK
```

Type `wchar_t` is the data type needed to handle wide characters. This data type is defined in `stdlib.h/std-def.h` as the type `unsigned short`.

## 4.2.2 Library Functions and Header Files

This package contains an ANSI standard library and an emulation library for calculating floating-point numbers and the remainders of divided integral numbers.

The header files in the "include" directory contain library function declarations and macro definitions. When using a library function, include the header file that contains its declaration by using the `#include` instruction.

The table below shows the relationship between the types of library files and the header files.

Table 4.2.2.1 List of library files and functions

### ANSI standard library

File name	Functions/macros	Corresponding header file
libc.a	tmpfile*, tmpnam*, remove*, rename*, fopen*, freopen, fclose*, setbuf*, setvbuf*, fflush*, clearerr*, feof*, ferror*, perror, fseek*, fgetpos*, fsetpos*, ftell*, rewind*, getchar, fgetc, getc, gets, fgets, fscanf, scanf, sscanf, fread, putchar, fputc, putc, puts, fputs, ungetc, fprintf, printf, sprintf, vfprintf, vsprintf, fwrite	stdio.h
	abort, exit, atexit*, getenv*, system*, malloc, calloc, realloc, free, atoi, atol, atof, strtol, strtoul, strtod, abs, labs, div, ldiv, rand, srand, bsearch, qsort	stdlib.h
	setjmp, longjmp	setjmp.h
	time, difftime*, clock*, mktime, localtime*, gmtime, asctime*, ctime*	time.h
	acos, asin, atan, atan2, ceil, cos, cosh, exp, fabs, floor, fmod, frexp, ldexp, log, log10, modf, pow, sin, sinh, sqrt, tan, tanh	math.h, errno.h, float.h, limits.h
	memchr, memmove, strchr, strcspn, strncat, strpbrk, strstr, memcmp, memset, strcmp, strerror, strncmp, strrchr, strtok, memcpy, strcat, strcpy, strlen, strncpy, strspn	string.h
	isalnum, iscntrl, isgraph, isprint, isspace, isxdigit, toupper, isalpha, isdigit, islower, ispunct, isupper, tolower	ctype.h
	va_start, va_arg, va_end	stdarg.h

The functions marked with an asterisk (\*) are dummy functions.

### Emulation library

File name	Functions
libgcc.a	__adddi3, __subdi3, __adddf3, __addsf3, __ashldi3, __ashldi3, __ashlsi3, __ashrdi3, __ashrhi3, __ashrsi3, __cmpdi2, __divdf3, __divdi3, __divhi3, __divsf3, __divsi3, __eqdf2, __eqsf2, __extendsfdf2, __fcmpd, __fcmps, __ffsdi2, __fixdfdi, __fixdfsi, __fixsfdi, __fixsfsi, __fixunssfdi, __fixunssfsi, __fixunssfsi, __floatdidf, __floatdisf, __floatsidf, __floatsisf, __gedf2, __gesf2, __gtdf2, __gtsf2, __ledf2, __lesf2, __lshrdi3, __lshrhi3, __lshrsi3, __ltdf2, __ltsf2, __moddi3, __modhi3, __modsi3, __muldf3, __muldi3, __mulhi3, __mulsf3, __mulsi3, __nedf2, __negdf2, __negdi2, __negsf2, __nesf2, __subdf3, __subsf3, __truncdfsf2, __ucmpdi2, __udivdi3, __udivhi3, __udivsi3, __umoddi3, __umodhi3, __umodsi3

For details about the functions included in the libraries, refer to Chapter 7, "Library".

When using a library function, be sure to specify the library file that contains the function used when linking. The linker extracts only the necessary object modules from the specified library file as it links them.

### 4.2.3 In-line Assemble

The **xgcc** C compiler supports in-line assembly, so the **asm** statement can be used. As a result, the word "asm" is reserved for system use.

Format: **asm ("*character string*") ;**

Example 1: 

```
/* HALT mode */
asm("halt");
```

Example 2: 

```
/* Trap Table*/
asm(".long      BOOT\n\
     .long      ADDR_ERR\n\
     .long      NMI\n\
     .space     4\n\
     .long      EINT0\n\
     .long      EINT1");
```

Example 3: 

```
BOOT() {
    asm("xld.a  %sp,0x3f00"); /* set SP */
    :
}
```

For details on how to write an assembly source, refer to Section 4.3, "Grammar of Assembly Source".

### 4.2.4 Prototype Declarations

#### Declaring interrupt handler functions

Interrupt handler functions should be declared in the following format:

**<type> <function name> \_\_attribute\_\_ ((interrupt\_handler));**

Example: 

```
void foo(void) __attribute__ ((interrupt_handler));
```

```
int int_num;
void foo()
{
    int_num = 5;
}
```

#### Assembler code

```
foo:
    ld.a  -[%sp],%r2
    ld    %r2,5
    xld   [int_num],%r2
    ld.a  %r2,[%sp]+
    reti
```

## 4.3 Grammar of Assembly Source

### 4.3.1 Statements

Each individual instruction or definition of an assembly source is called a statement. The basic composition of a statement is as follows:

#### Syntax pattern

1	<Mnemonic>	(<Operands>)	(;<Comment>)
2	<Assembler directive>	(<Parameters>)	(;<Comment>)
3	<Label>:		(;<Comment>)
4	;<Comment>		
5	<Extended instruction>	<Operands>	(;<Comment>)
6	<Preprocessor directive>	(<Parameters>)	(;<Comment>)

Example:

	Statement	Syntax pattern
	; boot.s	4
	; boot program	4
	#define SP_INI,0x3f00 ; Stack pointer value	6
	.text	2
	.long BOOT ; BOOT VECTOR	2
	BOOT:	3
	xld.a %sp,SP_INI ; set SP	5
	xcall main ; goto main	5
	jpr BOOT ; infinity loop	1
	:	:

The example given above is an ordinary source description method. For increased visibility, the elements composing each statement are aligned with tabs and spaces.

#### Restrictions

- Only one statement can be described in one line. A description containing more than two instructions in one line will result in an error. However, comments may be described in the same line with an instruction or label.

Example: ;OK

```
BOOT:    ld  %r1,%r2
         ld  %r0,%r1
;Error
BOOT:    ld  %r1,%r2          ld  %r0,%r1
```

- One statement cannot be described in more than one line. A statement not complete in one line will result in an error.

Example: ;OK

```
         ld  %r1,%r2
;Error
         ld  %r1,
         %r2
```

- The usable characters are limited to ASCII characters (alphanumeric symbols), except for use in comments. Also, the usable symbols have certain limitations (details below).

Comments can be described using other characters than ASCII characters. When using non-ASCII characters (such as Chinese characters) for comments, use /\* . . . \*/ as the comment symbol.

## (1) Instructions (Mnemonics and Operands)

An instruction to the S1C17 Core is generally composed of *<Mnemonic>* + *<Operand>*. Some instructions do not contain an operand.

### General notation forms of instructions

General forms: *<Mnemonic>*

*<Mnemonic>* tab or space *<Operand>*

*<Mnemonic>* tab or space *<Operand 1>*, *<Operand 2>*

Examples:    nop  
              call    SUB1  
              ld     %r0, 0x4

There is no restriction as to where the description of a mnemonic may begin in a line. A tab or space preceding a mnemonic is ignored. Generally, mnemonics are justified left by tab setting.

An instruction containing an operand needs to be broken with one or more tabs or spaces between the mnemonic and the operand. If there are plural operands, the operands are separated from each other with one comma (.). Space between operands is ignored.

The elements of operands will be described further below.

### Types of mnemonics

The following S1C17 Core instructions can be used in the S1C17 Family:

ld.b	ld.ub	ld	ld.a		
add	add/c	add/nc	add.a	add.a/c	add.a/nc
adc	adc/c	adc/nc	sub	sub/c	sub/nc
sub.a	sub.a/c	sub.a/nc	sbc	sbc/c	sbc/nc
cmp	cmp/c	cmp/nc	cmp.a	cmp.a/c	cmp.a/nc
cmc	cmc/c	cmc/nc			
and	and/c	and/nc	or	or/c	or/nc
xor	xor/c	xor/nc	not	not/c	not/nc
sr	sa	sl	swap		
cv.ab	cv.as	cv.al	cv.la	cv.ls	
jpr	jpr.d	jpa	ipa.d	jrgt	jrgt.d
jrge	jrge.d	jrlt	jrlt.d	jrle	jrle.d
jrugt	jrugt.d	jruga	jruga.d	jrult	jrult.d
jrle	jrle.d	jreq	jreq.d	jrne	jrne.d
call	call.d	calla	calla.d	ret	ret.d
int	intl	reti	reti.d	brk	ret.d
ext	nop	halt	slp	ei	di
ld.cw	ld.ca	ld.cf			

Refer to the "S1C17 Core Manual" for details of each instruction.

### Restrictions on characters

Mnemonics can be written in uppercase (A–Z) characters, lowercase (a–z) characters, or both. For example, "ld", "LD", and "Ld" are all accepted as "ld" instructions.

For purposes of discrimination from symbols, this manual uses lowercase characters.

More will be said about operands later.



## (2) Assembler Directives

The `as` assembler supports the standard directives provided in the gnu assembler. Refer to the gnu assembler manual for the standard directives. Each directive begins with a period (.). The following lists often-utilized directives.

<code>.text</code>		Declares a <code>.text</code> section.
<code>.section .data</code>		Declares a <code>.data</code> section.
<code>.section .rodata</code>		Declares a <code>.rodata</code> section.
<code>.section .bss</code>		Declares a <code>.bss</code> section.
<code>.long</code>	<code>&lt;data&gt;</code>	Defines a 4-byte data.
<code>.short</code>	<code>&lt;data&gt;</code>	Defines a 2-byte data.
<code>.byte</code>	<code>&lt;data&gt;</code>	Defines a byte data.
<code>.ascii</code>	<code>&lt;string&gt;</code>	Defines an ASCII character strings.
<code>.space</code>	<code>&lt;length&gt;</code>	Defines a blank (0x0) space.
<code>.zero</code>	<code>&lt;length&gt;</code>	Defines a blank (0x0) space.
<code>.align</code>	<code>&lt;value&gt;</code>	Alignment to a specified boundary address.
<code>.global</code>	<code>&lt;symbol&gt;</code>	Defines a global symbol.
<code>.set</code>	<code>&lt;symbol&gt;, &lt;address&gt;</code>	Defines a symbol with an absolute address.

## (3) Labels

A label is an identifier designed to refer to an arbitrary address in the program. You can refer to a branch destination of a program or an address in the `.text/.data` section by using a symbol defined as a label.

### Definition of a label

A symbol described in the following format is regarded as a label.

`<Symbol>:`

Preceding spaces and tabs are ignored. It is a general practice to describe from the top of a line.

A defined symbol denotes the address of a described location.

An actual address value will be determined in the linking process.

### Restrictions

Only the following characters can be used:

A-Z a-z \_ 0-9

A label cannot begin with a numeral. Uppercase and lowercase are discriminated.

Examples: `;OK`                    `;Error`  
`FOO:`                    `llabel:`  
`_Abcd:`                `0_ABC:`  
`L1:`

## (4) Comments

Comments are used to describe the meaning of a series of routines or each statement. Comments cannot comprise part of coding.

### Definition of comment

A character string beginning with a semicolon (;) and ending with a line feed is interpreted as a comment.

Strings from "/\*" through the next "\*/" are also regarded as a comment.

Not only ASCII characters, but also other non-ASCII characters can be used to describe a comment.

It can be described with a label or instruction in one line.

Examples: ; This line is a comment line.

```

LABEL:                ;Comment for LABEL.
    ld    %a,%b      ;Comment for the instruction on the left.

/*
    This type of comment can include
    newline characters.
*/

```

### Restrictions

When a comment extends to several lines, each line must begin with a semicolon or use "/\*" and "\*/".

Examples:

```

;These are
  comment lines.    The second line will not be regarded as a comment. An error will result.

;These are
;  comment lines.  Both lines will be regarded as comments.

/*
  These are
  comment lines.    Both lines will be regarded as a comment.
*/

```

## (5) Blank Lines

This assembler also allows a blank line containing only a return/line feed code. It need not be made into a comment line, for example, when used as a break in a series of routines.

## 4.3.2 Notations of Operands

This section explains the notations for the register names, symbols, and constants that are used in the operands of instructions.

### (1) Register Names

The names of the internal registers of the S1C17 Core all contain a percentage symbol (%). Register names may be written in either uppercase or lowercase letters.

General-purpose register (%rd, %rs, %rb)	Notation
General-purpose register R0–R7	%r0–%r7 or %R0–%R7
Special register	Notation
Stack pointer SP	%sp or %SP
Program counter PC	%pc or %PC

Register names placed in brackets ([]) for indirect addressing must include the % symbol.

Examples: [%r7] [%r1]+ [%sp+imm7]

**Note:** A register name not containing % will be regarded as a symbol.

Conversely, all notations beginning with % will be regarded as registers, and will give rise to an error if it is not a register name.

### (2) Numerical Notations

The as assembler supports three kinds of numerical notations: decimal, hexadecimal and binary.

#### Decimal notations of values

Notations represented with 0–9 only will be regarded as decimal numbers. To specify a negative value, put a minus sign (-) before the value.

Examples: 1 255 -3

Characters other than 0–9 and the sign (-) cannot be used.

#### Hexadecimal notations of values

To specify a hexadecimal number, place "0x" before the value.

Examples: 0x1a 0xff00

"0x" cannot be followed by characters other than 0–9, a–f, and A–F.

#### Binary notations of values

To specify a binary number, place "0b" before the value.

Examples: 0b1001 0b01001100

"0b" cannot be followed by characters other than 0 or 1.

#### Specified ranges of values

The size (specified range) of immediate data varies with each instruction.

The specifiable ranges of different immediate data are given below.

Table 4.3.2.1 Types of immediate data and their specifiable ranges

Symbol	Type	Decimal	Hexadecimal	Binary
imm3	3-bit immediate data	0 to 7	0x0 to 0x7	0b0 to 0b111
imm5	5-bit immediate data	0 to 31	0x0 to 0x1f	0b0 to 0b1 1111
imm7	7-bit immediate data	0 to 127	0x0 to 0x7f	0b0 to 0b111 1111
sign7	Signed 7-bit immediate data	-64 to 63	0x0 to 0x7f	0b0 to 0b111 1111
sign8	Signed 8-bit immediate data	-128 to 127	0x0 to 0xff	0b0 to 0b1111 1111
sign10	Signed 10-bit immediate data	-512 to 511	0x0 to 0x3ff	0b0 to 0b11 1111 1111
imm13	13-bit immediate data	0 to 8,191	0x0 to 0x1fff	0b0 to 0b1 1111 1111 1111
imm16	16-bit immediate data	0 to 65,535	0x0 to 0xffff	0b0 to 0b1111 1111 1111 1111
sign16	Signed 16-bit immediate data	-32,768 to 32,767	0x0 to 0xffff	0b0 to 0b1111 1111 1111 1111
imm20	20-bit immediate data	0 to 1,048,575	0x0 to 0xfffff	0b0 to 0b1111 1111 1111 1111 1111
sign21	Signed 21-bit immediate data	-1,048,576 to 1,048,575	0x0 to 0x1ffff	0b0 to 0b1 1111 1111 1111 1111 1111
sign23	Signed 23-bit immediate data	-4194304 to 4194303	0x0 to 0x7ffff	0b0 to 0b111 1111 1111 1111 1111 1111
imm24	24-bit immediate data	0 to 16,777,215	0x0 to 0xfffff	0b0 to 0b1111 1111 1111 1111 1111 1111
sign24	Signed 24-bit immediate data	-8,388,608 to 8,388,607	0x0 to 0xfffff	0b0 to 0b1111 1111 1111 1111 1111 1111

### (3) Symbols

In specifying an address with immediate data, you can use a symbol defined in the source files.

#### Definition of symbols

Usable symbols are defined as 24-bit values by any of the following methods:

1. It is described as a label (in text, data or bss section)

Example: LABEL1:

LABEL1 is a symbol that indicates the address of a described location in the `.text`, `.data`, or `.bss` section.

2. It is defined with the `.set` directive

Example: `.set ADDR1, 0xff00`

ADDR1 is a symbol that represents absolute address 0x00ff00.

#### Restrictions on characters

The characters that can be used are limited to the following:

A-Z a-z \_ 0-9

Note that a symbol cannot begin with a numeral. Uppercase and lowercase characters are discriminated.

#### Local and global symbols

Defined symbols are normally local symbols that can only be referenced in the file where they are defined.

Therefore, you can define symbols with the same name in multiple files. To reference a symbol defined in some other file, you must declare it to be global in the file where the symbol is defined by using the `.global` directive.

#### Extended notation of symbols

When referencing an address with a symbol, you normally write the name of that symbol in the operand where an address is specified.

Examples: `call LABEL ← LABEL = sign10`  
`ld.a %rd, LABEL ← LABEL = sign7`

The `as` assembler also accepts the referencing of an address with a specified displacement as shown below.

`LABEL + imm24 LABEL + sign24`

Example: `xcall LABEL+0x10`

### 4.3.3 Extended Instructions

The extended instructions are such that the contents which normally are written in multiple instructions including the `ext` instruction can be written in one instruction. Extended instructions are expanded into the smallest possible basic instructions by the `as` assembler.

#### Types of extended instructions

<code>xadd</code>	<code>xadd.a</code>	<code>xadc</code>	<code>xsub</code>	<code>xsub.a</code>	<code>xsbc</code>	<code>xcmp</code>
<code>xcmp.a</code>	<code>xcmc</code>					
<code>sadd</code>	<code>sadd.a</code>	<code>sadc</code>	<code>ssub</code>	<code>ssub.a</code>	<code>ssbc</code>	<code>scmp</code>
<code>scmp.a</code>	<code>scmc</code>					
<code>xand</code>	<code>xoor</code>	<code>xxor</code>				
<code>sand</code>	<code>soor</code>	<code>sxor</code>				
<code>xld</code>	<code>xld.a</code>	<code>xld.b</code>	<code>xld.ub</code>			
<code>sld</code>	<code>sld.a</code>	<code>sld.b</code>	<code>sld.ub</code>			
<code>xjpr</code>	<code>xjpr.d</code>	<code>xjpa</code>	<code>xjpa.d</code>	<code>xjreq</code>	<code>xjreq.d</code>	<code>xjrne</code>
<code>xjrne.d</code>	<code>xjrgt</code>	<code>xjrgt.d</code>	<code>xjrge</code>	<code>xjrge.d</code>	<code>xjrnt</code>	<code>xjrnt.d</code>
<code>xjrle</code>	<code>xjrle.d</code>	<code>xjrnt</code>	<code>xjrnt.d</code>	<code>xjrge</code>	<code>xjrge.d</code>	<code>xjrnt</code>
<code>xjrnt.d</code>	<code>xjrle</code>	<code>xjrle.d</code>	<code>xcall</code>	<code>xcall.d</code>	<code>xcalla</code>	<code>xcalla.d</code>
<code>sjpr</code>	<code>sjpr.d</code>	<code>sjpa</code>	<code>sjpa.d</code>	<code>sjreq</code>	<code>sjreq.d</code>	<code>sjrne</code>
<code>sjrne.d</code>	<code>sjrnt</code>	<code>sjrnt.d</code>	<code>sjrge</code>	<code>sjrge.d</code>	<code>sjrnt</code>	<code>sjrnt.d</code>
<code>sjrle</code>	<code>sjrle.d</code>	<code>sjrnt</code>	<code>sjrnt.d</code>	<code>sjrge</code>	<code>sjrge.d</code>	<code>sjrnt</code>
<code>sjrnt.d</code>	<code>sjrle</code>	<code>sjrle.d</code>	<code>scall</code>	<code>scall.d</code>	<code>scalla</code>	<code>scalla.d</code>
<code>xld.cw</code>	<code>xld.ca</code>	<code>xld.cf</code>				
<code>sld.cw</code>	<code>sld.ca</code>	<code>sld.cf</code>				

#### Method for using extended instructions

The value or symbol for the expanded immediate size can be written directly in the operand.

```
Examples: xcall LABEL ; ext LABEL [23:10]
          ; call LABEL [9:0]

          sld.a %r1,imm16 ; ext imm16 [15:7]
          ; ld.a %r1,imm16 [6:0]

          xld.a %r1,imm24 ; ext imm24 [23:20]
          ; ext imm24 [19:7]
          ; ld.a %r1,imm24 [6:0]
```

In addition to the immediate expansion function of the basic instructions, a special operand specification like the one shown below is accepted for some instructions.

```
Examples: xld.a %r0,symbol + 0x10 ; R0 ← symbol + 0x10
          xjpa LABEL + 5 ; Jumps to address LABEL + 5.
```

For details about the extended instructions that include operands, refer to Section 8.6, "Extended Instructions".

### 4.3.4 Preprocessor Directives

The **cpp** C preprocessor directives can be used in assembly source files.

The principal directives are as follows:

<b>#include</b>	Insertion of file
<b>#define</b>	Definition of character strings and numbers
<b>#macro-#endm</b>	Definition of macros
<b>#if-#else-#endif</b>	Conditional assembly

```
Examples: #include "define.h"
          #define NULL 0
          #macro ADDM $1,$2
            xld %r0,[$1]
            xld %r1,[$2]
            add %r0,%r1
            xld [$1],%r0
          #endm
          #ifdef TYPE1
            ld %r0,0
          #else
            ld %r0,-1
          #endif
```

Refer to the gnu C preprocessor manual for details of the preprocessor directives.

**Note:** The sources that contain preprocessor directives need to be processed by the preprocessor (use the **xgcc** options `-c` and `-xassembler-with-cpp`), and cannot be entered directly into the **as** assembler. (Direct entry into the assembler will result an error.)

## 4.4 Precautions for Creation of Sources

- (1) Place a tab stop every 4 characters wherever possible. Source display/mixed display with the **gdb** debugger of a source set at a tab interval other than 4 characters may result in displaced output of the source part.
- (2) When compiling/assembling a C source or assembly source that includes debugging information, do not include other source files (by using `#include`). It may cause a debugger operation error. This does not apply to ordinary header files that do not contain sources.
- (3) When using C and assembler modules in a program, pay attention to the interface between the C functions and assembler routines, such as arguments, size of return values and the parameter passing conventions.
- (4) The C compiler assumes that the address size is 24 bits by default or 20 bits when the `-mshort-offset` option only is specified. Therefore, be aware that the expected results may not be obtained from an operation using an `unsigned int/unsigned short` type variable and a pointer, as `int` type variables are 16-bit size, as shown below.

```
int* ip_Pt;
unsigned int i = 1;

ip_Pt += (-1)*i;
```

The code above is written to expect `"ip_Pt += (-1);"`, however it will be processed as `"ip_Pt += 0xffff;"`.

Although it will be processed normally when the address space is 16-bit size, an invalid address will result if the address space is 24- or 20-bit size.

To perform a pointer operation correctly when the C compiler is under the default condition or when the `-mshort-offset` option only is specified, avoid using `unsigned int/unsigned short` type variables, or add the suffix `'L'` to the constant as shown below so that it will be handled as a `long` type constant.

```
ip_Pt += (-1L)*i;
```

- (5) In C sources, function names can be used as the pointer to the function, note, however, that the pointer values cannot be assigned to real type (`float/double`) variables and arrays using the function names.

They can be assigned to integer type variables and arrays.

However, if assigning a value to a global variable/array using a function name at the same time the variable/array is declared, the types of variables/arrays are limited depending on the address space size.

In 24-bit address space (default condition) or 20-bit address space (when `-mshort-offset` only is specified)

The `long/unsigned long` type variables/arrays only allow substitution with a function name.

In 16-bit address space (when `-mpointer16` is specified)

The `short/unsigned short/int/unsigned int` type variables/arrays only allow substitution with a function name.

If it is not at declaration, global variables/arrays in any integer type can be substituted with a function name.

Integer type local variables/arrays always allow substitution with a function name regardless of whether it is at declaration or not.

Examples: In 16-bit address space (when `-mpointer16` is specified)

```
1) short s_Global_Val = (short)boot;
```

→ A function name can be used to assign the pointer to the `short` type global variable `s_Global_Val` when it is declared.

```
2) long l_Global_Val = (long)boot;
```

→ An error occurs if a function name is assigned to the `long` type global variable `l_Global_Val` when it is declared.

```
error: initializer element is not constant
```

```
3) short s_local_val = (short)boot;
```

→ A function name can be used to assign the pointer to the `short` type local variable `s_local_val`.

## 4 SOURCE FILES

4) `long l_local_val = (long)boot;`  
→ A function name can be used to assign the pointer to the `long` type local variable `l_local_val`.

5) `char c_Global_Val;`

```
void sub()
{
    c_Global_Val = (char)boot;
}
```

→ A function name can be used to assign the pointer to the `char` type global variable `c_Global_Val` except when it is declared.

(6) Function pointers can be used in C sources, note, however, that function pointers cannot be assigned to real type (`float/double`) variables and arrays similar to (5) above.

They can be assigned to integer type variables and arrays.

However, when a global variable/array is declared, a function pointer cannot be assigned at the same time.

If it is not at declaration, a function pointer can be assigned to integer type global variables/arrays.

Integer type local variables/arrays always allow substitution with a function pointer regardless of whether it is at declaration or not. However, a warning occurs depending on a combination of the address space size and the type of global/local variable or global/local array.

In 24-bit address space (default condition) or 20-bit address space (when `-mshort-offset` only is specified)

A warning occurs if a function pointer is assigned to a variable/array other than `long/unsigned long` data types.

In 16-bit address space (when `-mpointer16` is specified)

A warning occurs if a function pointer is assigned to a variable/array other than `short/unsigned short/int/unsigned int` data types.

Examples: In 16-bit address space (when `-mpointer16` is specified)

```
void (* fp_Pt)(void);    // Declaration of a function pointer with
                        // void type return value and argument
```

1) `short s_Global_Val = (short)fp_Pt;`

→ An error occurs if a function pointer is assigned to the global variable `s_Global_Val` when it is declared.

```
error: initializer element is not constant
```

2) `short s_local_val = (short)fp_Pt;`

→ A function pointer can be assigned to the `short` type local variable `s_local_val`.

3) `long l_local_val = (long)fp_Pt;`

→ Although a function pointer can be assigned to the `long` type local variable `l_local_val`, a warning will occur.

```
warning: cast from pointer to integer of different size
```

4) `short s_Global_Val;`

```
void sub()
{
    s_Global_Val = (short)fp_Pt;
}
```

→ A function pointer can be assigned to the `short` type global variable `s_Global_Val` except when it is declared.



- (7) Be sure to declare the prototypes of the functions. Although the functions will be compiled without an error or warning if there is no prototype declaration, the return value is implicitly assumed to be an `int` type. Therefore, the correct value will not be returned if the return value is a larger data type than `int`.

Example:

```
long l_Val=0x12345678,l_Val_2;
```

```
int main()
```

```
{
    l_Val_2 = sub();    // l_Val_2 is substituted with 0x5678.
    return 0;
}
```

```
long sub()
```

```
{
    long l_wk;
    l_wk = l_Val;
    return l_wk;
}
```

THIS PAGE IS BLANK.

# 5 GNU17 IDE

This chapter describes the facilities available with the **GNU17 IDE** and describes how to use the **GNU17 IDE**.

## 5.1 Overview

---

### 5.1.1 Features

The **GNU17 IDE** (hereafter simply the **IDE**) provides an integrated development environment that makes it user to develop software using the S1C17 Family C Compiler Package (S5U1C17001C).

The main features of the **IDE** are outlined below.

- **Project management**  
Allows collective management of all source files needed to create an application as a single project.
- **Supports GNU-compliant C and assembler**  
The **IDE** lets users create and edit sources in GNU-compliant C or assembly language. User can also load source code written in other editors into the **IDE**.
- **Creates and executes a makefile**  
The **IDE** automatically generates a makefile featuring a compiler to linker execution sequence based on user-selected build options. A build process to generate an executable object file based on this file can be executed simply by clicking a button.
- **Creates various definition files**  
The **IDE** lets the user easily edit/create various files in addition to the above makefile, including the linker script file needed to build a project and the parameter and command files needed to launch the debugger.
- **Launcher for calling the **gdb** debugger**  
After a build process, the user can call the **gdb** debugger to debug a built application.

### 5.1.2 Some Notes on Use of the IDE

#### About the guaranteed operation of the IDE

The **IDE** is designed to run on the Eclipse development platform and uses Eclipse facilities during development work. Note that the facilities not described in this manual lie beyond the scope of guarantee for the **IDE**.

#### Eclipse plug-in versions

Listed below are the Eclipse plug-ins and versions required by the **IDE**:

Table 5.1.2.1 Eclipse plug-in versions

Plug-in	Version
Eclipse Platform	3.2.0
Eclipse CDT	3.1.0

**IDE** operations cannot be guaranteed if any modification, deletions, or updates of these plug-ins are made. Since the **IDE** is a Java application, Java Virtual Machine will be installed as well. **IDE** operations cannot be guaranteed for use on a platform other than this virtual Java machine.


#### About the use of Japanese language in the IDE

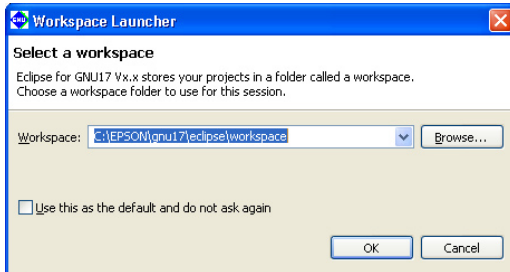
Although the **IDE** permits Japanese (using Shift-JIS/MS-932 character code) file and directory names and strings, the GNU17 tools used to build projects do not support the Japanese language. Do not use the Japanese language for file and directory names or in executable source code. (Comments in the source code may be written in Japanese.)

## 5.2 Starting and Quitting the IDE

### 5.2.1 Starting the IDE

The method for starting the **IDE** is described below.

- (1)  Double-click the **eclipse.exe** icon in the `c:\EPSON\gnu17\eclipse` directory to start the **IDE**. You can also start the **IDE** by selecting [EPSON MCU] > [GNU17] > [GNU17 IDE] from the Windows Start menu, or from the command line without parameters.
- (2) After the Eclipse splash screen, the [Workspace Launcher] dialog box shown below is displayed. Here, specify the working directory (workspace) in which you want to store projects and associated files.

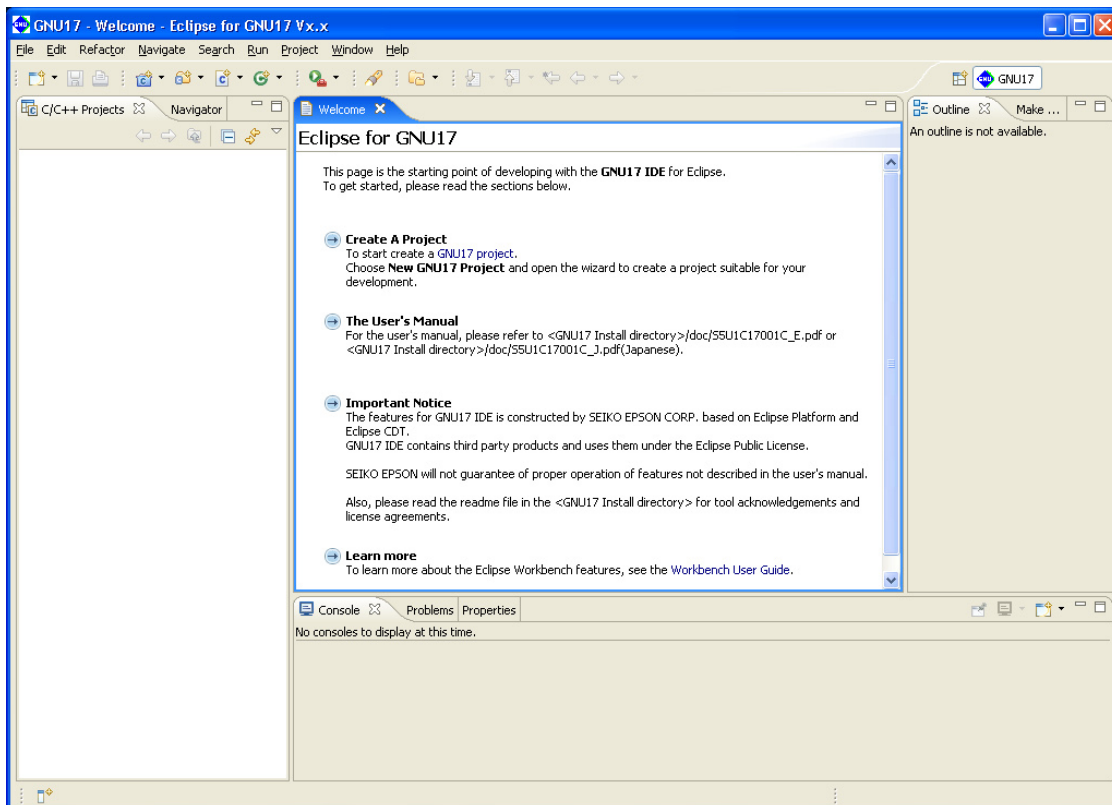


Although `c:\EPSON\gnu17\eclipse\workspace` is displayed as the default directory, you can select any other directory or create a new directory and set it as the workspace. Enter a directory name in the [Workspace:] combo box or select one from the directory select dialog box displayed by clicking the [Browse...] button.

The [Workspace Launcher] dialog box is displayed each time you start the **IDE**. If you plan to perform your work in the same workspace from this point, you can choose not to display the [Workspace Launcher] dialog box by selecting the [Use this as the default and do not ask again] check box (indicated by a check mark when selected). (Select [Switch Workspace...] from the [File] menu to change to a different workspace.)

- (3) Click the [OK] button.

The **IDE** window shown below will be displayed.

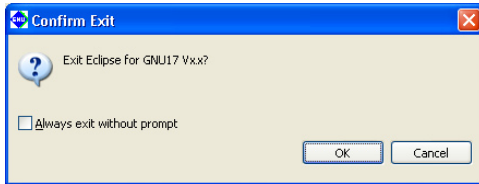


## 5.2.2 Quitting the IDE

Select [Exit] from the [File] menu to close the **IDE**.

If any open files in the editor have not been saved, you will be prompted to save or discard your changes. Select [Yes] or [No] before quitting the **IDE**.

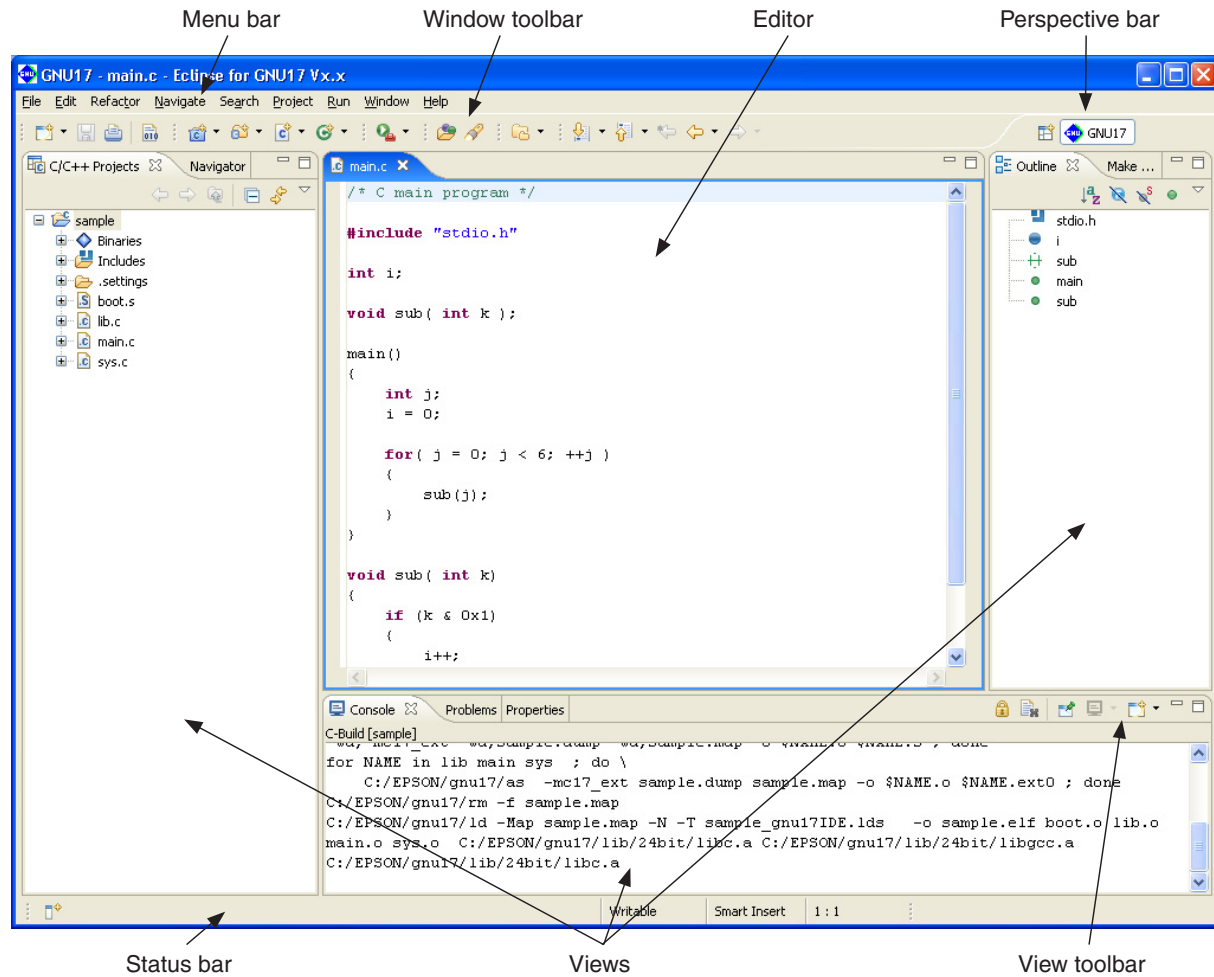
You also can use the  (close) button to quit the **IDE**. Click the [OK] button at the following dialog prompt to quit or [Cancel] to continue working.



Select the [Always exit without prompt] check box to skip this prompt.

## 5.3 IDE Window

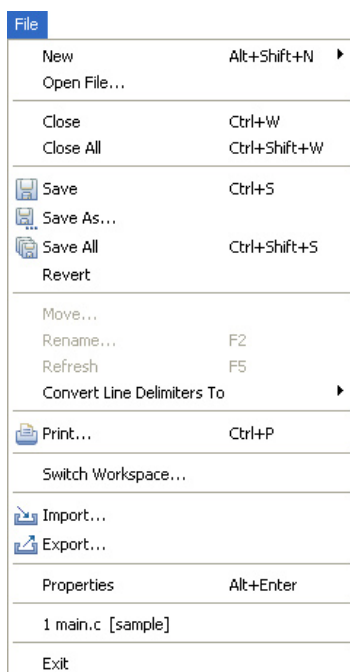
The IDE window consists of an editor surrounded by several views and a menu bar and a toolbar.



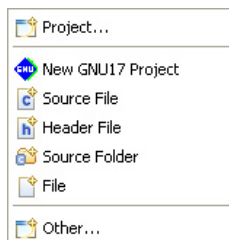
## 5.3.1 Menu Bar

File Edit Refactor Navigate Search Project Run Window Help

### [File] menu



### New (Alt+Shift+N)



### Project...

Selects a wizard to create a new project. Select [GNU17 Project] > [New GNU17 Project] from the various wizards shown in the displayed dialog box.

### New GNU17 Project

Launches a GNU17 wizard to create a new project.

### Source File

Creates a new source file.

### Header File

Creates a new header file.

### Source Folder

Creates a new source folder.

### File

Creates a new text file.

### Other...

Works the same way as [Project...].

### Open File...

Choose a file to be opened with the editor.

### Close (Ctrl+W)

Closes the current active file. You will be prompted to save or discard any changes made since you created or last saved the file.

### Close All (Ctrl+Shift+W)

Closes all files open in the editor. You will be prompted to save or discard any changes made since you created or last saved the files.

### Save (Ctrl+S)

Saves changes made in the current file. If the content you last edited has already been saved, selecting menu command has no effect.

### Save As...

Saves the current active file under another name or at a different location.

### Save All (Ctrl+Shift+S)

Saves all open files.

### Revert

Discards any changes made in the current active file, reverting to the previously saved version.

### Move...

Moves the file or directory selected in the [C/C++ Projects] or [Navigator] view to a different location.

### Rename... (F2)

Places the file or directory selected in the [C/C++ Projects] or [Navigator] view in editing mode (allowing renaming of the file or directory).

### Refresh (F5)

Updates the displayed content of the [C/C++ Projects] or [Navigator] view.

**Convert Line Delimiters To**

Selects a line delimiting character.

**Print... (Ctrl+P)**

Prints the current active file.

**Switch Workspace...**

Selects another workspace. All information for the current workspace is saved before the new workspace is opened. Open files are saved as if by the [Save All] command.

**Import...**

Launches a wizard that lets the user add an existing project or source file to the current workspace or project.

**Export...**

Writes the file in the current project out to another directory.









**Properties (Alt+Enter)**

Opens a dialog box in which the user can display or edit properties of the project, file, or directory currently selected in the [C/C++ Projects] or [Navigator] view.

**Exit**

Closes the IDE.

**[Edit] menu**

Edit	
 Undo Typing	Ctrl+Z
 Redo Typing	Ctrl+Y
<hr/>	
 Cut	Ctrl+X
 Copy	Ctrl+C
 Paste	Ctrl+V
<hr/>	
 Delete	Delete
Select All	Ctrl+A
<hr/>	
Find/Replace...	Ctrl+F
Find Next	Ctrl+K
Find Previous	Ctrl+Shift+K
Incremental Find Next	Ctrl+J
Incremental Find Previous	
<hr/>	
Add Bookmark...	
Add Task...	
<hr/>	
Word Completion	Alt+/
<hr/>	
 Shift Right	Ctrl+I
 Shift Left	Ctrl+Shift+I
Content Assist	Ctrl+Space
Add Include	Ctrl+Shift+N
Format	Ctrl+Shift+F
Open On Selection	
<hr/>	
Set Encoding...	

**Undo Typing (Ctrl+Z)**

Undoes the most recent operation performed in the editor.

**Redo Typing (Ctrl+Y)**

Repeats the last operation canceled by [Undo Typing].

**Cut (Ctrl+X)**

Cuts the selected string or file/directory and copies it to the clipboard.

**Copy (Ctrl+C)**

Copies a selected string or file/directory to the clipboard.

**Paste (Ctrl+V)**

Pastes the copied content from the clipboard to the position indicated by the cursor or into the current view.

**Delete (Delete)**

Deletes the selected string or file/directory.

**Select All (Ctrl+A)**

Selects all contents in the currently active editor.

**Find/Replace... (Ctrl+F)**

Finds and replaces a string in the editor.

**Find Next (Ctrl+K)**

Jumps to the next instance of a search string.

**Find Previous (Ctrl+Shift+K)**

Jumps back to the previous instance of a search string.

**Incremental Find Next (Ctrl+J)**

Select this command and type a string to search for the string in the currently active document (searched backward from the current cursor position). This command performs another search each time you type one character and jumps to the next instance of the current search string when you press the arrow keys [↑] or [↓]. You can cancel this search mode by pressing the arrow keys [←] or [→] or the [Enter] or [Esc] key.



### Incremental Find Previous

Select this command and type a string to search for the string in the currently active document (searched forward from the current cursor position). This command performs another search each time you type one character and jumps to the next instance of the current search string when you press the arrow keys [ $\uparrow$ ] or [ $\downarrow$ ]. You can cancel this search mode by pressing the arrow keys [ $\leftarrow$ ] or [ $\rightarrow$ ] or the [Enter] or [Esc] key.

### Add Bookmark...

Registers a line in an active document in the editor at the current cursor position as a bookmark. For more information on the bookmarks, refer to Section 5.5.6, "Bookmarks".

### Add Task...

Registers the line at the current cursor position in an active document in the editor as a task (memorandum). The registered task can be managed in the [Tasks] view.

### Word Completion (Alt+/)

Inserts a word that begins with the letters being entered in the editor to the current position. The most recently entered word is selected.

### Shift Right (Ctrl+I)

Shifts the beginning of a line one tab position to the right.

### Shift Left (Ctrl+Shift+I)

Shifts the beginning of a line one tab position to the left.

### Content Assist (Ctrl+Space)

Displays a dialog box and inserts a C source reserved word or template selected in the dialog box at the current cursor position. This command is available only when editing C source code.

### Set Encoding...

Selects the text-encoding format.

## [Refactor] menu

Refactor	
Undo	Alt+Shift+Z
Redo	Alt+Shift+Y
Rename...	Alt+Shift+R

### Undo (Alt+Shift+Z)

Undoes the most recent rename operation.

### Redo (Alt+Shift+Y)

Repeats the rename operation canceled by [Undo].

### Rename... (Alt+Shift+R)

Changes the selected function or member name, all instances including these in other locations of the source.

## [Navigate] menu

Navigate	
Go Into	
Go To	
Open Type...	Ctrl+Shift+T
Show In	Alt+Shift+W
Next Annotation	Ctrl+.
Previous Annotation	Ctrl+,
Last Edit Location	Ctrl+Q
Go to Line...	Ctrl+L
Back	Alt+Left
Forward	Alt+Right

### Go Into

Changes display of the [C/C++ Projects] or [Navigator] view to display the content of just the currently selected directory.

### Go To

Back
Forward
Up One Level

### Back

Reverts to the immediately preceding [C/C++ Projects] or [Navigator] view.

### Forward

Switches from the above view to the subsequent state.

### Up One Level

Switches the display of the [C/C++ Projects] or [Navigator] view to display the content one level above the current hierarchical level.

### Show In

Selects a view other than the editor (if available) to highlight the resource that includes the selected element (function name, variable name, or type).

**Next Annotation (Ctrl+.)**

Selects the next item the list displayed in the [Problems] or the [Search] view.

**Previous Annotation (Ctrl+,)**

Selects the previous item the list displayed in the [Problems] or the [Search] view.

**Last Edit Location (Ctrl+Q)**

Jumps to the last edited position in the editor.

**Go to Line... (Ctrl+L)**

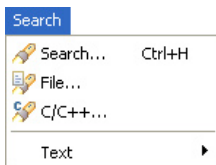
Jumps to the position in the active document indicated by the specified line number.

**Back (Alt+Left)**

Returns to any position in the document just referenced or edited.

**Forward (Alt+Right)**

Reverts the display traced back by [Back] above to the next recent state.

**[Search] menu****Search... (Ctrl+H)**

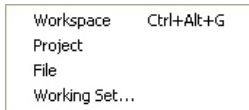
Displays a [Search] dialog box that lets the user search for a file or C.

**File...**

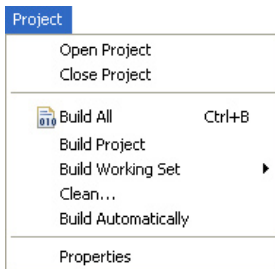
Searches for a file containing the specified string. (Displays the [Search] dialog box file search page.)

**C/C++...**

Searches for C source containing the specified string. (Displays the C search page of the [Search] dialog box.)

**Text**

Searches the string at which the cursor is currently placed within the range (work space, current project, current file, or specified working set) selected from the submenu.

**[Project] menu****Open Project**

Opens the closed project currently selected in the [C/C++ Projects] or [Navigator] view.

**Close Project**

Closes the project currently selected in the [C/C++ Projects] or [Navigator] view.

**Build All (Ctrl+B)**

Executes a build process on all projects open in the [C/C++ Projects] or [Navigator] view.

**Build Project**

Executes a build process on the project currently selected in the [C/C++ Projects] or [Navigator] view.

**Build Working Set**

Executes a build process on the resources included in a specified working set.

**Clean...**

Deletes all files that were generated during the previous build process to repeat a build process from all resources.

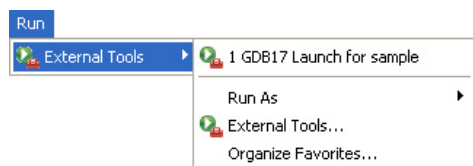
**Build Automatically**

Turns the auto-build feature on or off. This feature allows the user to automatically execute a build after saving source files edited in the editor, but cannot be used within the **IDE**.

## Properties

Displays a [Properties] dialog box that lets the user display or edit properties of the project selected in the [C/C++ Projects] or [Navigator] view.

## [Run] menu



### External Tools

#### Tool list

Lists the external tool names set or selected with [External Tools...].

External tools can be executed by selecting in this list.

#### External Tools...

Makes the settings required to start the **gdb** debugger or an external tool. You also can start a tool from the displayed dialog box. Once launched, the tool is displayed in the tool list.

#### Organize Favorites...

Registers frequently used external tools.

## [Window] menu



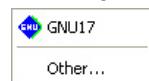
### New Window

Opens a new window in the initially set view layout of the currently selected perspective. The currently open project is moved unchanged to the new window view.

### New Editor

Opens the currently edited file with the new editor tab.

### Open Perspective



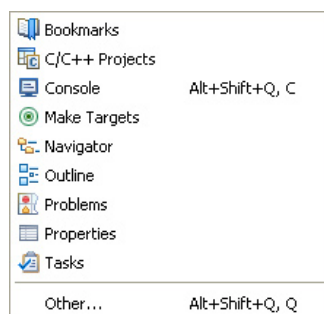
#### GNU17

Opens the GNU17 perspective.

#### Other...

Opens another perspective.

### Show View



Opens the view selected in a submenu. If the view is already open, the view is activated.

### Customize Perspective...

Allows the user to make changes to toolbar shortcuts or settings for menu commands defined in the current perspective.

### Save Perspective As...

Saves settings for the current perspective under another name.

### Reset Perspective

Restores the perspective (view layout, etc.) to the default state.

### Close Perspective

Closes the currently active perspective.

### Close All Perspectives

Closes all loaded perspectives.

## Navigation

Show System Menu	Alt+-
Show View Menu	Ctrl+F10
Maximize Active View or Editor	Ctrl+M
Minimize Active View or Editor	
Activate Editor	F12
Next Editor	Ctrl+F6
Previous Editor	Ctrl+Shift+F6
Switch to Editor...	Ctrl+Shift+E
Quick Switch Editor	Ctrl+E
Next View	Ctrl+F7
Previous View	Ctrl+Shift+F7
Next Perspective	Ctrl+F8
Previous Perspective	Ctrl+Shift+F8

### Show System Menu (Alt+-)

Displays the currently active view or system menus usable in the editor (e.g., fast view, resize, or close).

### Show View Menu (Ctrl+F10)

Displays view menus for the currently active view.

### Maximize Active View or Editor (Ctrl+M)

Maximizes the view or editor of the currently active view. If already maximized, the view or editor reverts to the original size.

### Minimize Active View or Editor

Minimizes the view or editor of the currently active view.

### Activate Editor (F12)

Activates the document displayed in front of all other documents currently open in the editor.

### Next Editor (Ctrl+F6)

Selects the document to be activated in the editor (by default, the one opened just after the currently active document in usage history).

### Previous Editor (Ctrl+Shift+F6)

Selects the document to be activated in the editor (by default, the one opened just before the currently active document in usage history).

### Switch to Editor... (Ctrl+Shift+E)

Selects the document to be activated in the editor from the dialog box that appears.

### Quick Switch Editor (Ctrl+E)

Selects the document to be activated in the editor from the pull-down list that appears.

### Next View (Ctrl+F7)

Selects the view to be activated (by default, the one opened just after the current view in usage history).

### Previous View (Ctrl+Shift+F7)

Selects the view to be activated (by default, the one opened just before the current view in usage history).

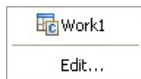
### Next Perspective (Ctrl+F8)

Selects the perspective to be activated (by default, the one opened just after the currently active perspective in usage history).

### Previous Perspective (Ctrl+Shift+F8)

Selects the perspective to be activated (by default, the one opened just before the currently active perspective in usage history).

## Working Sets



### List of working sets

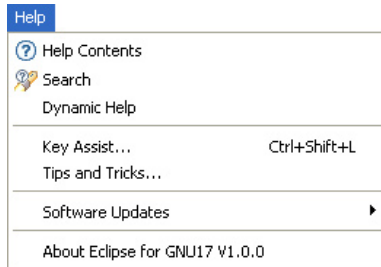
Lists the working sets that have been created. Select a working set to be processed from the list.

### Edit...

Creates, edits, or deletes a working set.

## Preferences...

Displays a [Preferences] dialog box that lets users customize the **IDE** environment.

**[Help] menu****Help Contents**

Displays help contents in a browser.

**Search**

Displays a search view for help topics.

**Dynamic Help**

Displays the help topic related to the view currently activated.

**Key Assist... (Ctrl+Shift+L)**

Displays the list of currently available menu commands.

**Software Updates**

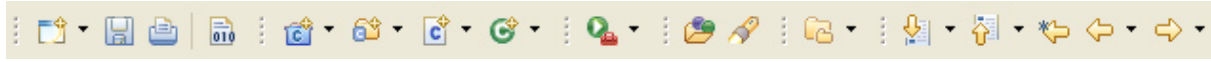
Installs an updater, updates, plug-ins, etc. for software management.

Use this command only when required.









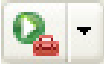



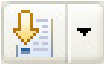


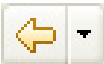
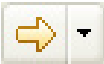
**About Eclipse for GNU17 Vx.x**

Shows **IDE** version information and detailed information on plug-ins, etc.

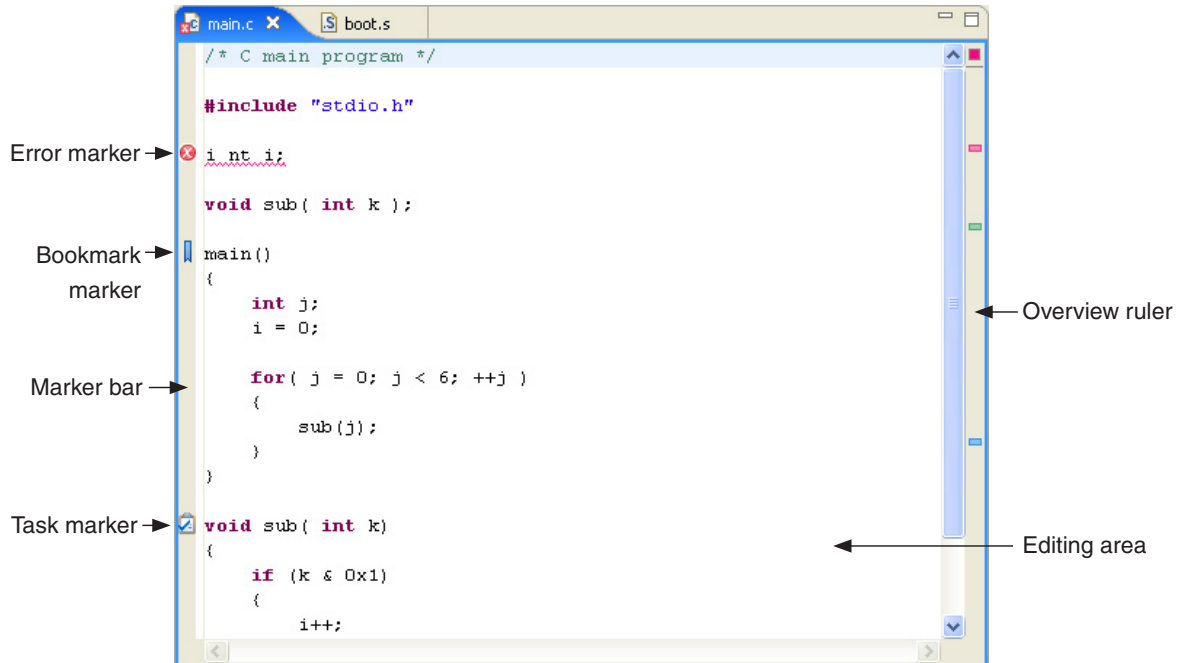
### 5.3.2 Window Toolbar



This toolbar contains shortcuts to frequently used commands from the window menu. For information on each button, refer to the description of the menu bar in the preceding section.

	New	= [File] > [New]
	Save	= [File] > [Save]
	Print	= [File] > [Print...]
	Build All	= [Project] > [Build All]
	New C/C++ Project	= [File] > [New] > [New GNU17 Project]
	New C/C++ Source Folder	= [File] > [New] > [Source Folder]
	New C/C++ Source File	= [File] > [New] > [Source File]
	New C++ Class	Unused
	External Tools (Run GDB17 Launch for xxxxx)	= [Run] > [External Tools]
	Open Type	Unused
	Search	= [Search] > [Search...]
	Select Working Sets	= [Window] > [Working Sets]
	Next Annotation	= [Navigate] > [Next Annotation]
	Previous Annotation	= [Navigate] > [Previous Annotation]
	Last Edit Location	= [Navigate] > [Last Edit Location]
	Back	= [Navigate] > [Back]
	Forward	= [Navigate] > [Forward]

### 5.3.3 Editor Area



This is the area of the editor where you edit source code. The **IDE** has an editor for C sources and an editor for assembler sources. These editors have the same features as a general-purpose editor, and error messages or variable or function names displayed in other views can be linked to the editor. Multiple documents can be opened at a time, any of which can be selected with a tab at the top of the area in which its document name is displayed.

The marker bar on the left edge of the editor area shows the line in error and the markers indicating a bookmark, a line in which a task is set, etc. Hover the mouse pointer over a marker to display the contents of an error, the name of a bookmark, or a task explanation.

As for the marker bar, the overview ruler on the right edge of the area shows the position in error and the position at which a bookmark or task is set by a square symbol. The positions displayed on this side do not correspond to the current display position; they are relative positions seen from the entire file. Hover the mouse pointer over a symbol to display explanations as for the marker. Click a symbol to go to that position.

In addition to the built-in editors, you can start an external editor from the **IDE** and edit the sources in it. For more information on editing features, refer to Section 5.5, "The Editor and Editing of Source Files".

## Context menu

Right-click in the editing area to display the context menu shown below.

Undo Typing	Ctrl+Z
Revert File	
Save	
Show In	Alt+Shift+W ▶
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Shift Right	
Shift Left	
Comment	Ctrl+/ /
Uncomment	Ctrl+\
Add Block Comment	Ctrl+Shift+/ /
Remove Block Comment	Ctrl+Shift+\
Content Assist	Ctrl+Space
Add Include	
Format	
Show in C/C++ Projects	
Refactor	▶
Open Declaration	F3
Open Definition	Ctrl+F3
Go to next member	Ctrl+Shift+Down
Go to previous member	Ctrl+Shift+Up
Declarations	▶
References	▶
Search Text	▶
Run As	▶
Debug As	▶
Team	▶
Compare With	▶
Replace With	▶
Object file conversion	▶
Preferences...	
Create Make Target...	
Build Make Target...	

**Undo Typing** = [Edit] > [Undo Typing]

### Revert File

Restores the document being editing to the content previously saved to the document file.

**Save** = [File] > [Save]

### Show In

Activates the view ([C/C++ Projects] view, [Navigator] view, or [Outline] view) selected from the submenu and highlights the file being currently edited. (C editor only)

**Cut** = [Edit] > [Cut]

**Copy** = [Edit] > [Copy]

**Paste** = [Edit] > [Paste]

**Shift Right** = [Edit] > [Shift Right]

**Shift Left** = [Edit] > [Shift Left]

### Comment

Changes the line at which the cursor is currently located to a comment line (by adding "//" to the beginning of the line). Even when the current line is already a comment, the "//" is added. (C editor only)

### Uncomment

Changes the comment line at which the cursor is currently located to an ordinary source line (by deleting "//" from the beginning of the line). If the line is not preceded by "//", this command has no effect, even if the line is enclosed in a set of "/\* \*/". (C editor only)

### Add Block Comment

Encloses the character string/lines with "/\* \*/" to comment out. (C editor only)

**Content Assist** = [Edit] > [Content Assist] (C editor only)

### Show in C/C++ Projects

Activates the [C/C++ Projects] view (opened even when it is closed or minimized), in which the function or variable name at the current line indicated by the cursor if any is highlighted. (C editor only)

### Refactor

Rename...

Changes the selected type, function, or member name, all instances including these in other locations of the source. After this operation, an [Undo .....] or [Redo .....] command is displayed in a submenu, letting you cancel or re-execute the operation.

### Go to next member

Jumps to the immediately following location where a function or variable is defined. (C editor only)

### Go to previous member

Jumps to the immediately preceding location where a function or variable is defined. (C editor only)

### Declarations

Searches the location where the string selected in the editing area (e.g., function name or variable name) is declared within the range selected in the submenu (workspace, current project, specified working set). (C editor only)



**References**

Searches the location where the string selected in the editing area (e.g., function name or variable name) is referenced within the range selected in the submenu (workspace, current project, specified working set). (C editor only)

**Search Text**

Searches the location where the string selected in the editing area is referenced within the range selected in the submenu (workspace, current project, specified working set). (C editor only)

**Preferences...**

Displays the preference dialog box for the editor.

**Create Make Target...**

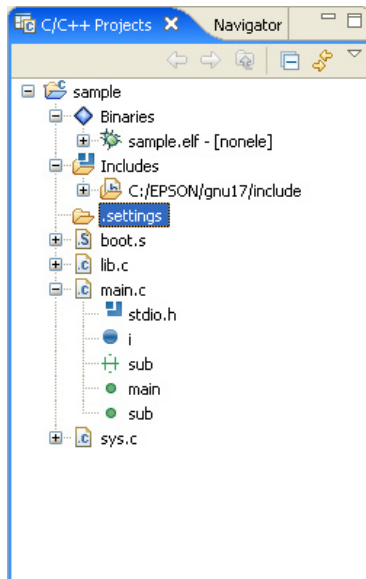
Defines the target to be selected by [Build Make Target...].

**Build Make Target...**

Selects a target and executes **make.exe** on it.

(For the menu commands not specifically discussed here, refer to the description of the menu bar.)

### 5.3.4 [C/C++ Projects] View



Lists the projects present in the workspace along with the C and assembler sources, include files, and generated execution format object files included in these projects. (Select the type of file to be displayed using [Filters...] from the view menu.) The function names and global variable names, etc. in the C source can also be displayed. Before editing a project or source or performing other operations, be sure to select the desired project or source here.

The file list displayed in tree structure can be navigated in the same way as with Windows Explorer.

Display the contents of a directory/file or fold them up into the parent directory by clicking the or icon. To display the content of only a specific directory, select the desired directory and then [Go Into] from the menu. To redo, click the [Up] button in the toolbar shown below.

Navigation operations are saved to a history file, and the operations can be restored to a previous state or advanced forward using the [Back] or [Forward] menu command or toolbar button.

**Note:** For assembler sources, this view may not always display correctly.

#### Tree list icons

Indicated below are the meanings of the main icons displayed in the tree list.

Project	Include
Binary container	Variable
Executable format file	Function
Include container	Structure (struct)
Include folder	Member variable
Header file	Union (union)
Source folder	Enumeration type (enum)
C source file	Enumerator
Assembler source file	Function definition (prototype declaration)
Text file, etc.	Macro-definition
Object file	Type definition (typedef)

#### Toolbar



##### Back

Restores the display in the view to the immediately preceding state based on history.



##### Forward

Advances the display in the view to the immediately following state based on history.



##### Up

Expands the display in the view to the immediately higher hierarchy.

**Collapse All**

Folds all of the hierarchy-expanded display ([-]) up into the uppermost hierarchy ([+]).

**Link with Editor**

While this button is toggled, the editor view changes to reflect the selected content in the view. For example, when you select (click) a file in the view, the selected document is displayed in front of all other documents in the editor (providing the editor is already open). When you select a C source function name or variable name displayed in the view, the editor view jumps to the beginning of the function or the position at which the variable is defined.

**Menu****Select Working Set...**

Selects, creates, or deletes a working set. A working set is used to limit the resources to be displayed to a specific view.

**Deselect Working Set**

Restores a selected working set to an unselected state.

**Edit Active Working Set...**

Edits the content of the currently selected working set.

**Filters...**

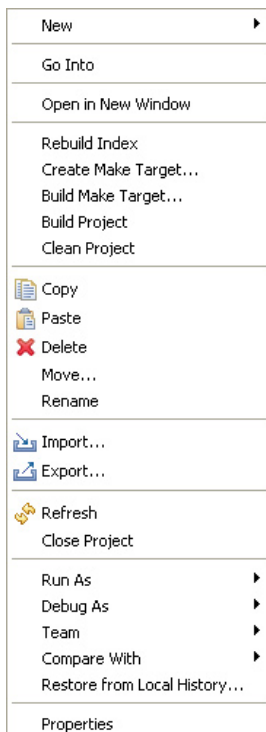
Specifies the type of file to be displayed.

**Link With Editor**

Updates the editor view to reflect the selection in the view.

**Context menu**

Right-click in the view to display the context menu shown below.



**New** = [File] > [New]

**Go Into** = [Navigate] > [Go Into]  
(Effective when a project is selected)

**Open in New Window** = [Window] > [New Window]  
(Effective when a project is selected)

**Open**  
Opens a selected file in the editor. (Effective when a file is selected)

**Open With**  
Opens a selected file in the editor currently selected in the submenu shown below. (Effective when a file is selected)

**C/C++ Editor (Assembly Editor)**

C editor (when C source is selected) or assembly editor (when assembler source is selected)

**Text Editor**

Text editor

**System Editor**

Windows program (e.g., Notepad)

**In-Place Editor**

C editor (when C source is selected) or assembly editor (when assembler source is selected)

**Default Editor**

C editor (when C source is selected) or assembly editor (when assembler source is selected)

**Create Make Target...**

Defines the target to be selected by [Build Make Target...].

**Build Make Target...**

Selects a target and executes **make.exe** on it.

**Build Project** = [Project] > [Build Project] (Effective when a project is selected)

**Clean Project**

Executes "Clean" described in the makefile. (Effective when a project is selected)

**Copy** = [Edit] > [Copy]

**Paste** = [Edit] > [Paste]

**Delete** = [Edit] > [Delete]

**Move...** = [File] > [Move...]

**Rename** = [File] > [Rename...]

**Import...** = [File] > [Import...]

**Export...** = [File] > [Export...]

**Refresh** = [File] > [Refresh]

**Close Project** = [Project] > [Close Project] (Effective when a project is selected)

**Add Bookmark...** = [Edit] > [Add Bookmark...] (Effective when a file is selected)

**Compare With**

Each Other

Compares the contents of two or three selected files with each other.

Local History...

Compares the content of a selected file with its previously saved content. (Effective when a file is selected)

**Restore from Local History...**

Restores files (such as these that have been deleted) back in the project. (Effective when a project is selected)

**Replace With** (Effective when a file is selected)

Previous from Local History

Replaces a selected file with the content that was saved immediately before.

Local History...

Replaces a selected file with its previously saved content (selected from history).

**Object file conversion** (Effective when a file is selected)

Generate an S record file (Effective when an elf file is selected)

Converts the selected elf format object file into Motorola S3 format to generates a HEX file.

This command executes "objcopy -O srec --srec-forceS3 <file name>.elf <file name>.sa".

Generate a raw binary file (Effective when an elf file is selected)

Removes debugging and other information from the selected elf format object file to generates a binary file.

This command executes "objdump -O binary <file name>.elf <file name>.bin".

**Properties**

Displays a [Properties] dialog box that lets the user display or change properties of the current project.

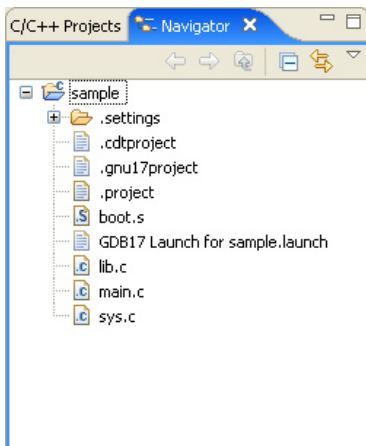
**Refactor** (Effective when a file is selected)

Rename...

Changes the selected type, function, or member name, all instances including these in other locations of the source. After this operation, an [Undo .....] or [Redo .....] command is displayed in a submenu, letting you cancel or re-execute the operation.



(For the menu commands not specifically discussed here, refer to the description of the menu bar.)

### 5.3.5 [Navigator] View



Lists the directories and files present in the workspace. (The type of file to be displayed can be selected using [Filters...] from the view menu.) Before editing a project or source or performing other operations, select the desired project or source here.

The file list displayed in tree structure can be navigated in the same way as with Windows Explorer.

Display the contents of a directory/file or fold them up into only the parent directory by clicking the  or  icon. To display the content of only a specific directory, select the desired directory and then [Go Into] from the menu. To redo, click the [Up] button in the toolbar shown below.

Navigation operations are saved to a history file, and the operations can be restored to a previous state or advanced forward using the [Back] or [Forward] menu command or toolbar button.

#### Toolbar



##### Back

Restores the display in the view to the immediately preceding state based on history.



##### Forward

Advances display in the view to the immediately following state based on history.





##### Up

Expands the display in the view to the hierarchy one level up.



##### Collapse All

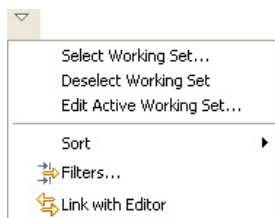
Folds all of the hierarchy-expanded display () up into the uppermost hierarchy ()



##### Link with Editor

While this button is toggled, the editor view changes to reflect the selected content in the view. For example, when you select (click) a file in the view, the selected document is displayed in front of all other documents in the editor (providing the editor is already open).

#### Menu



##### Select Working Set...

Selects, creates, or deletes a working set. A working set is used to limit the resources to be displayed to a specific view.

##### Deselect Working Set

Restores a selected working set to an unselected state.

##### Edit Active Working Set...

Edits the content of the currently selected working set.

##### Sort

###### by Name

Sorts display in the view in alphabetical order irrespective of file types.

###### by Type

Sorts display in the view in alphabetical order by file type.

##### Filters...

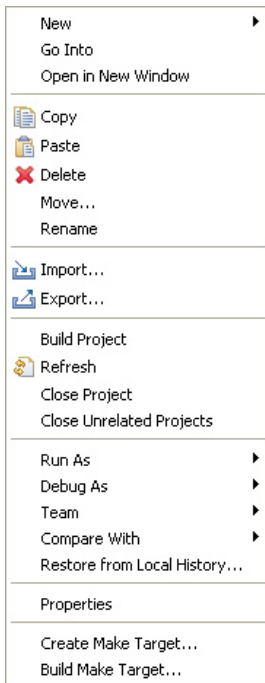
Specifies the type of file to be displayed.

##### Link with Editor

Updates the editor view to reflect the selection in the view.

## Context menu

Right-click in the view to display the context menu shown below.



**New** = [File] > [New]

**Go Into** = [Navigate] > [Go Into]  
(Effective when a project is selected)

**Open in New Window** = [Window] > [New Window]  
(Effective when a project is selected)

**Open**  
Opens a selected file in the editor. (Effective when a file is selected)

**Open With**  
Opens a selected file in the editor currently selected in the submenu shown below. (Effective when a file is selected)

**C/C++ Editor (Assembly Editor)**  
C editor (when C source is selected) or assembly editor (when assembler source is selected)

**Text Editor**  
Text editor

**System Editor**  
Windows program (e.g., Notepad)

**In-Place Editor**  
C editor (when C source is selected) or assembly editor (when assembler source is selected)

**Default Editor**  
C editor (when C source is selected) or assembly editor (when assembler source is selected)

**Copy** = [Edit] > [Copy]

**Paste** = [Edit] > [Paste]

**Delete** = [Edit] > [Delete]

**Move...** = [File] > [Move...]

**Rename** = [File] > [Rename...]

**Import...** = [File] > [Import...]

**Export...** = [File] > [Export...]

**Build Project** = [Project] > [Build Project] (Effective when a project is selected)

**Refresh** = [File] > [Refresh]

**Close Project** = [Project] > [Close Project] (Effective when a project is selected)

**Close Unrelated Projects**  
Closes the projects unrelated to the one being currently selected. (Effective when a project is selected)

**Compare With**  
Each Other  
Compares the contents of two or three selected files with each other.

Local History...  
Compares the content of a selected file with its previously saved content. (Effective when a file is selected)

**Restore from Local History...**  
Restores files (such as these that have been deleted) back in the project. (Effective when a project is selected)

**Replace With** (Effective when a file is selected)**Previous from Local History**

Replaces a selected file with the content that was saved immediately before.

**Local History...**

Replaces a selected file with its previously saved content (selected from history).

**Object file conversion** (Effective when a file is selected)**Generate an S record file** (Effective when an elf file is selected)

Converts the selected elf format object file into Motorola S3 format to generates a HEX file.

This command executes "objcopy -O srec --srec-forceS3 <file name>.elf <file name>.sa".

**Generate a raw binary file** (Effective when an elf file is selected)

Removes debugging and other information from the selected elf format object file to generates a binary file.

This command executes "objdump -O binary <file name>.elf <file name>.bin".

**Properties**

Displays a [Properties] dialog box that lets the user display or change properties of the current project.

**Create Make Target...**

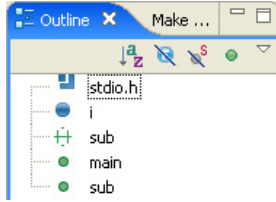
Defines the target to be selected by [Build Make Target...].

**Build Make Target...**

Selects a target and executes **make.exe** on it.

(For the menu commands not specifically discussed here, refer to the description of the menu bar.)

### 5.3.6 [Outline] View



Shows the functions and global variables that are written in the C source being displayed in the editor. Clicking on one of these items allows you to jump to the position in the editor at which the function or variable is written. While an assembler source is being displayed, no information is shown in this view.

The icons in the tree list are the same as in the [C/C++ Projects] view.

#### Toolbar



##### Sort

While this button is toggled, the displayed contents are sorted in alphabetical order. Normally, contents are displayed in the order in which they appear in the editor.



##### Hide Field

While this button is toggled, fields are not displayed.



##### Hide Static Members

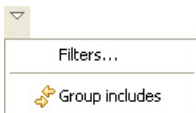
While this button is toggled, static members are not displayed.



##### Hide Non-Public Members

While this button is toggled, members other than public are not displayed.

#### Menu



##### Filters...

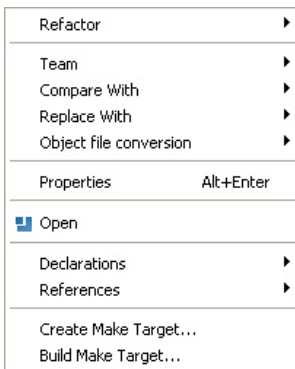
Specifies the items to be displayed in the view.

##### Group includes

Selects whether the included files are displayed in grouped structure or individually.

#### Context menu

Right-click in the view to display the context menu shown below.



##### Refactor

###### Rename...

Changes the selected type, function, or member name, all instances including these in other locations of the source. After this operation, an [Undo .....] or [Redo .....] command is displayed in a submenu, letting you cancel or re-execute the operation you performed.

##### Compare With

###### Each Other

Compares the contents of two or three selected files with each other.

###### Local History...

Compares the content of a selected file with its previously saved content.

##### Replace With

###### Previous from Local History

Replaces a selected file with the content that was saved immediately before.

###### Local History...

Replaces a selected file with its previously saved content (selected from history).

##### Object file conversion

This submenu is not used in this view.



**Properties**

Displays a [Properties] dialog box that lets the user display or change properties of the selected item.

**Open**

Opens a selected file in the editor. (Effective when a file is selected)

**Declarations**

Searches the location where the function name or variable name selected in the view is declared within the range selected in the submenu (workspace, current project, specified working set).

**References**

Searches the location where the function name or variable name selected in the view is referenced within the range selected in the submenu (workspace, current project, specified working set).

**Create Make Target...**

Defines the target to be selected by [Build Make Target...].

**Build Make Target...**

Selects a target and executes **make.exe** on it.

## 5.3.7 [Console] View

```

C-Build [sample]
C:\EPSON\gnu17\make.exe -f sample_gnu17IDE.mak all
C:/EPSON/gnu17/xgcc -BC:/EPSON/gnu17/ -c -xassembler-with-cpp -Wa,--gstabs -o boot.o
boot.s
C:/EPSON/gnu17/xgcc -BC:/EPSON/gnu17/ -gstabs -S -O1 -IC:/EPSON/gnu17/include -fno-builtin
-o lib.ext0 lib.c
C:/EPSON/gnu17/as -o lib.o lib.ext0
C:/EPSON/gnu17/xgcc -BC:/EPSON/gnu17/ -gstabs -S -O1 -IC:/EPSON/gnu17/include -fno-builtin

```

Displays the executed command line or the messages output by the GNU17 tools.

### Toolbar



#### Scroll Lock

While this button is toggled, automatic scroll is disabled.



#### Clear Console

Clears the contents displayed.



#### Pin Console

While this button is toggled, you can activate another view in the same pane even when a message is being output in the [Console] view. This button will prove useful when building a project takes time.



#### Display Selected Console

When multiple consoles such as a build console and a debugger startup console are open, this button allows you to select the console to be displayed in the [Console] view.



#### Open Console

Opens a new console.



#### Terminate

This button is displayed in a debugger startup console, etc. If you click this button, the tool corresponding to the console (e.g., the debugger) aborts the process underway and is closed. The console is not closed. Nor is the console closed when processing is terminated by an operation on the tool side.



#### Remove Launch

This button is displayed in a debugger startup console, etc. It closes the active console.

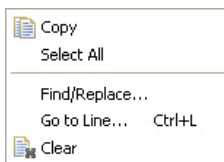


#### Remove All Terminated Launches

This button is displayed in a debugger startup console, etc. It closes all consoles of the terminated tools.

### Context menu

Right-click in the view to display the context menu shown below.



**Copy** = [Edit] > [Copy]

**Select All** = [Edit] > [Select All]

**Find/Replace...** = [Edit] > [Find/Replace...]

**Go to Line...**

Jumps to a specified line in the view.

#### Clear

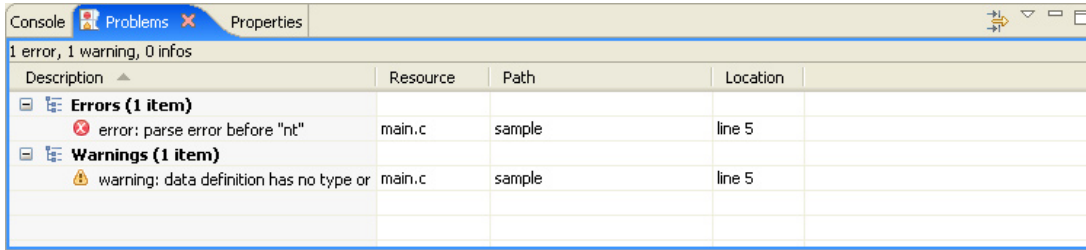
Clears the contents displayed.

#### Terminate

Displayed in the context menu of a console at debugger startup, etc. This menu command works the same way as the [Terminate] button described above.

(For the menu commands not specifically discussed here, refer to the description of the menu bar.)

## 5.3.8 [Problems] View



Shows the errors that occurred during a build operation. For errors in the source file, you can jump to the corresponding spot in the editor that is in error by clicking on an error message here.

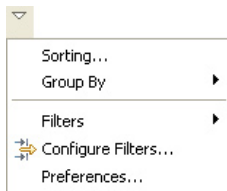
### Toolbar



#### Configure Filters

Displays a dialog box, letting you set conditions for the errors displayed.

### Menu



#### Sorting...

Sorts the errors displayed in list form according to the conditions you set in a dialog box.

#### Group By

Selects an error display format.

#### Filters

Selects a filter to be applied.

#### Configure Filters...

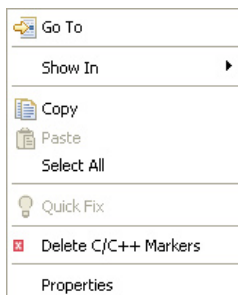
Same as the button described above.

#### Preferences...

Sets the maximum number of errors to be displayed.

### Context menu

Right-click in the view to display the context menu shown below.



#### Go To

Jumps to the line in the editor that is in error.

#### Show In

Selects a view other than the editor (if available) to highlight the resource in which the selected error has occurred.

**Copy** = [Edit] > [Copy]

**Select All** = [Edit] > [Select All]

#### Delete C/C++ Markers

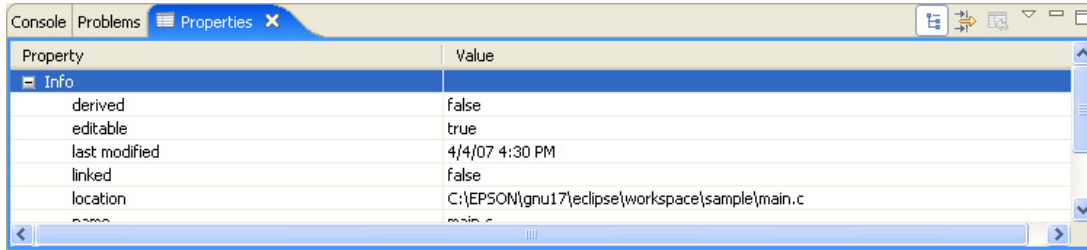
Deletes the error markers in the editor that correspond to the selected error.

#### Properties

Displays information on the error currently selected.

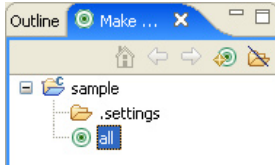
(For the menu commands not specifically discussed here, refer to the description of the menu bar.)

### 5.3.9 [Properties] View



Displays information on the resource or member currently selected in the [C/C++ Projects], the [Navigator], or the [Outline] view.

### 5.3.10 [Make Targets] View



When using a makefile you created, define the target here before executing it.

#### Toolbar



##### Home

Returns to the uppermost hierarchy in the tree list.



##### Back

Returns to the hierarchy one level up in the tree list.



##### Go Into

Advances to the hierarchy one level down in the tree list.



##### Build Make Target

Executes a make process on a selected target.

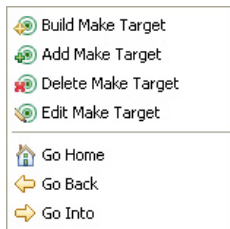


##### Hide Empty Folders

Hides the folders and displays registered targets only.

#### Context menu

Right-click in the view to display the context menu shown below.



##### Build Make Target

Executes a make process on a selected target.

##### Add Make Target

Defines a make target.

##### Delete Make Target

Deletes the selected target.

##### Edit Make Target

Edits a selected target.

##### Go Home

Returns to the uppermost hierarchy in the tree list.

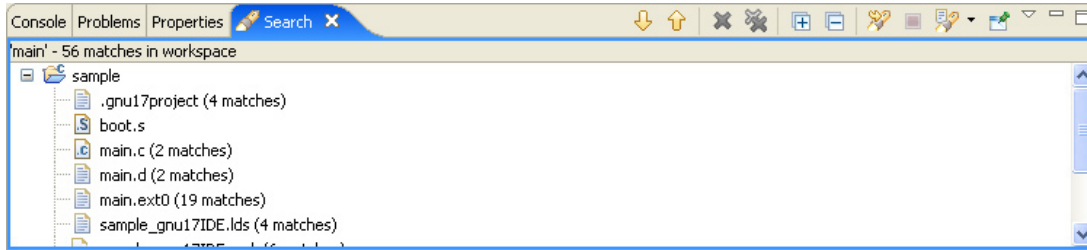
##### Go Back

Returns to the hierarchy one level up in the tree list.

##### Go Into

Advances to the hierarchy one level down in the tree list.

### 5.3.11 [Search] View



Shows the result of a search that was performed using the [Search] dialog box. This view in the initial IDE configuration is not displayed. It appears when a search is executed.

#### Toolbar



##### Show Next Match

Jumps to the next instance of search string immediately following the found occurrence.



##### Show Previous Match

Jumps to the previous instance of search string immediately preceding the found occurrence.



##### Remove Selected Matches

Deletes the found occurrence that you selected.



##### Remove All Matches

Deletes all of the found occurrences.



##### Expand All

Expands all of the hierarchical display in the view.



##### Collapse All

Folds all of the expanded hierarchical display up into the uppermost hierarchy.



##### Run the Current Search Again

Repeats the search previously performed.



##### Cancel Current Search

Cancels the search operation currently in progress.



##### Show Previous Searches

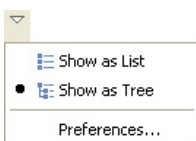
Shows the result of the previously performed search that you selected.



##### Pin the Search View

While this button is toggled, you can activate another view in the same pane even when the search results are being output in the [Search] view. This button will prove useful when a search takes time.

#### Menu



##### Flat Layout

Shows the search results in a non-hierarchical flat layout.

##### Hierarchical Layout

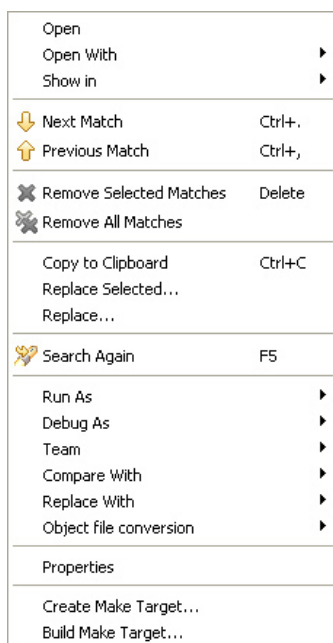
Shows the search results in hierarchically structured mode.

##### Preferences...

Displays the preference dialog box to set search conditions.

## Context menu

Right-click in the view to display the context menu shown below.



### Open

Opens a selected file in the editor. (Effective when searching a file)

### Open With

Opens a selected file in the editor currently selected in the submenu shown below. (Effective when searching a file)

### C/C++ Editor (Assembly Editor)

C editor (when C source is selected) or assembly editor (when assembler source is selected)

### Text Editor

Text editor

### System Editor

Windows program (e.g., Notepad)

### In-Place Editor

C editor (when C source is selected) or assembly editor (when assembler source is selected)

### Default Editor

C editor (when C source is selected) or assembly editor (when assembler source is selected)

### Show in

Highlights a selected occurrence of search string in the view selected from the submenu.

### Next Match

Jumps to the next instance of search string immediately following the found occurrence.

### Previous Match

Jumps to the previous instance of search string immediately preceding the found occurrence.

### Remove Selected Matches

Deletes the found occurrences of search string from the view that you selected.

### Remove All Matches

Deletes all of the found occurrences of search string from the view.

### Copy to Clipboard

Copies selected content to the clipboard.

### Replace Selected...

Replaces only the currently selected occurrence of search string with another string. (Effective when searching a file)

### Replace...

Replaces all of the currently selected occurrences of search string with another string. (Effective when searching a file)

### Search Again

Repeats the search previously performed.

### Compare With

#### Each Other

Compares the contents of two or three selected files with each other.

#### Local History...

Compares the content of a selected file with its previously saved content. (Effective when a file is selected)

**Restore from Local History...**

Restores files (such as these that have been deleted) back in the project. (Effective when a project is selected)

**Replace With** (Effective when a file is selected)**Previous from Local History**

Replaces a selected file with the content that was saved immediately before.

**Local History...**

Replaces a selected file with its previously saved content (selected from history).

**Object file conversion** (Effective when a file is selected)**Generate an S record file** (Effective when an elf file is selected)

Converts the selected elf format object file into Motorola S3 format to generates a HEX file.

This command executes "objcopy -O srec --srec-forceS3 <file name>.elf <file name>.sa".

**Generate a raw binary file** (Effective when an elf file is selected)

Removes debugging and other information from the selected elf format object file to generates a binary file.

This command executes "objdump -O binary <file name>.elf <file name>.bin".

**Properties**

Displays information on the currently selected occurrence of search string.

**Create Make Target...**

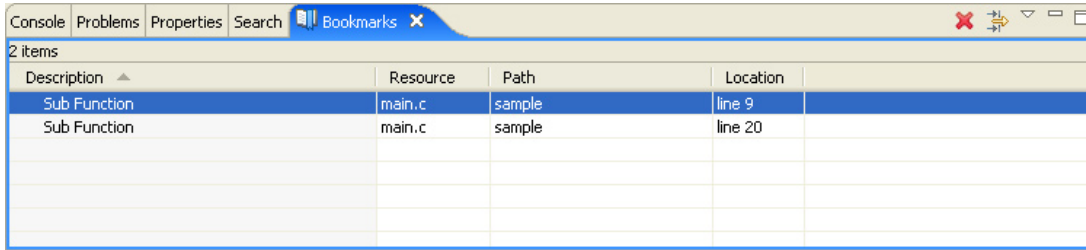
Defines the target to be selected by [Build Make Target...].

**Build Make Target...**

Selects a target and executes **make.exe**.



## 5.3.12 [Bookmarks] View



Shows the bookmarks registered in the editor, letting you jump to a bookmark or delete a bookmark. This view is not displayed in the initial **IDE** configuration. (You must select it by selecting [Show View] from the [Window] menu.)

### Toolbar



#### Delete

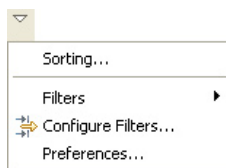
Deletes the selected bookmark.



#### Configure Filters

Sets conditions to display bookmarks.

### Menu



#### Sorting...

Sorts the bookmarks displayed in list form.

#### Filters

Selects a filter to be applied.

#### Configure Filters...

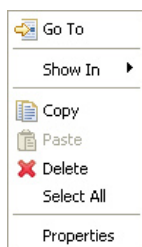
Same as the button described above.

#### Preferences...

Sets the maximum number of bookmarks to be displayed.

### Context menu

Right-click in the view to display the context menu shown below.



#### Go To

Jumps to a bookmark position in the editor.

#### Show In

Highlights the resource in which the selected bookmark is defined in the view selected from the submenu.

#### Copy = [Edit] > [Copy]

#### Paste = [Edit] > [Paste]

#### Delete = [Edit] > [Delete]

#### Select All = [Edit] > [Select All]



#### Properties

Displays information on the selected bookmarks.

(For the menu commands not specifically discussed here, refer to the description of the menu bar.)

### 5.3.13 [Tasks] View

Description	Resource	Path	Location
✓ Add Sub1 Function	main.c	sample	line 19
□ Modify Statement	main.c	sample	line 16
□ Check Performance	main.c	sample	line 25

Shows the tasks registered in the editor, letting you jump to or delete a task. A task is a "To-Do" item. The square  at the beginning of each line is the icon checked up on the completion of a task. The icon indicating priority (High = , Normal = blank, or Low = ) is displayed in the column next to the square. This view in the initial IDE configuration is not displayed. To open it, you must select it from [Show View] on the [Window] menu.

#### Toolbar



**Add Task** = [Edit] > [Add Task...]



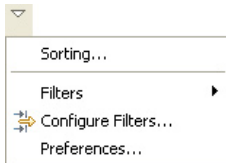
**Delete** = [Edit] > [Delete]



**Configure Filters**

Sets conditions to display tasks.

#### Menu



Sorting...

Sorts the tasks displayed in list form.

Filters

Selects a filter to be applied.

Configure Filters...

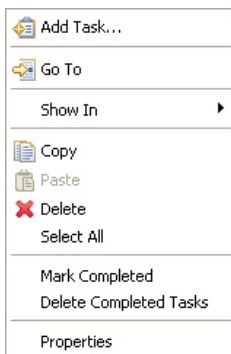
Same as the button described above.

Preferences...

Sets the maximum number of tasks to be displayed.

#### Context menu

Right-click in the view to display the context menu shown below.



**Add Task...** = [Edit] > [Add Task...]

**Go To**

Jumps to the position in the editor at which a task is set.

**Show In**

Highlights the resource in which a selected task is defined in the view selected from the submenu.

**Copy** = [Edit] > [Copy]

**Paste** = [Edit] > [Paste]

**Delete** = [Edit] > [Delete]

**Select All** = [Edit] > [Select All]

#### Mark Completed

Adds a completion mark to a selected task.

#### Delete Completed Tasks

Deletes all of the completed tasks.

#### Properties


Displays information on the tasks selected.

(For the menu commands not specifically discussed here, refer to the description of the menu bar.)

### 5.3.14 View Manipulation

This section describes how to open or close any view of the **IDE** and how to change the layout of a view.

#### Opening/closing a view

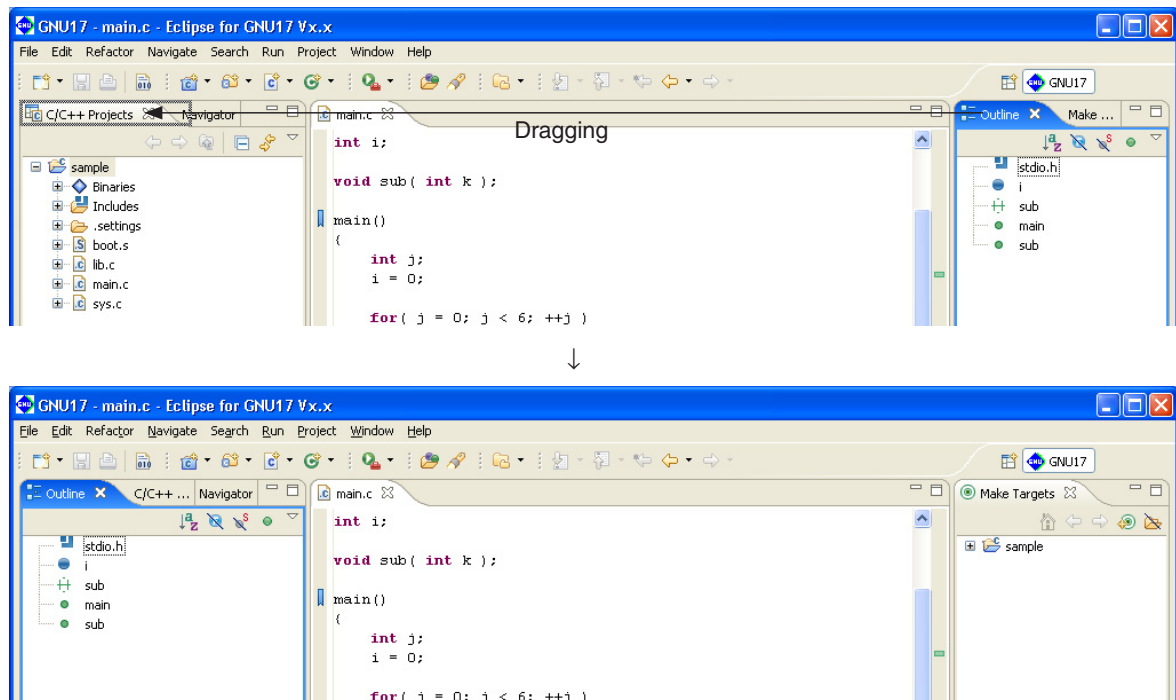
The displayed view is closed by clicking the  button on the tab. When all views in one pane are closed, the pane itself goes out.

To open a closed view, select it from [Show View] on the [Window] menu. The pane in which a selected view is displayed depends on how the perspective (described later) is set.

If multiple views overlap one on top of another in one pane, use the tab at the top of each view to select the view you want to display.

#### Changing the view layout

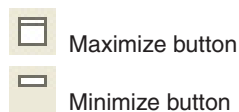
You can change the position at which a view is displayed by dragging its tab. When you drag the tab of a view to a relocatable position, a rectangular frame is displayed indicating the destination to which the view will be moved. For example, when you drag the tab of a view to a position in another pane and a frame in size of that pane and directory icons are displayed, the view is moved to that pane. Even when a frame in size of the tab is displayed at the tab position, the view is moved to that pane, in which case you can select a position in the stack of tabs at which you want to insert. If an arrow icon and a different size frame appears when you dragged a view's tab, the pane will be separated and the view will be displayed in a new pane.



The size of any pane can also be changed by dragging the boundary border of the pane.

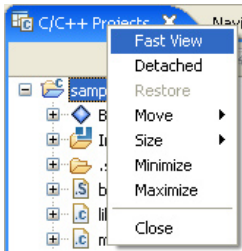
#### Maximizing a view

When you double-click the tab of a view, a pane including the selected view is expanded to the size of the **IDE** window, with other views hidden behind it. When you double-click the tab of a maximized view, the view reverts to its original size. Each pane has a maximize button at the upper right corner. Click this button to maximize a view. A minimize button restores the view to its original size. If you click the minimize button of a view in ordinary display, only the tab is displayed.



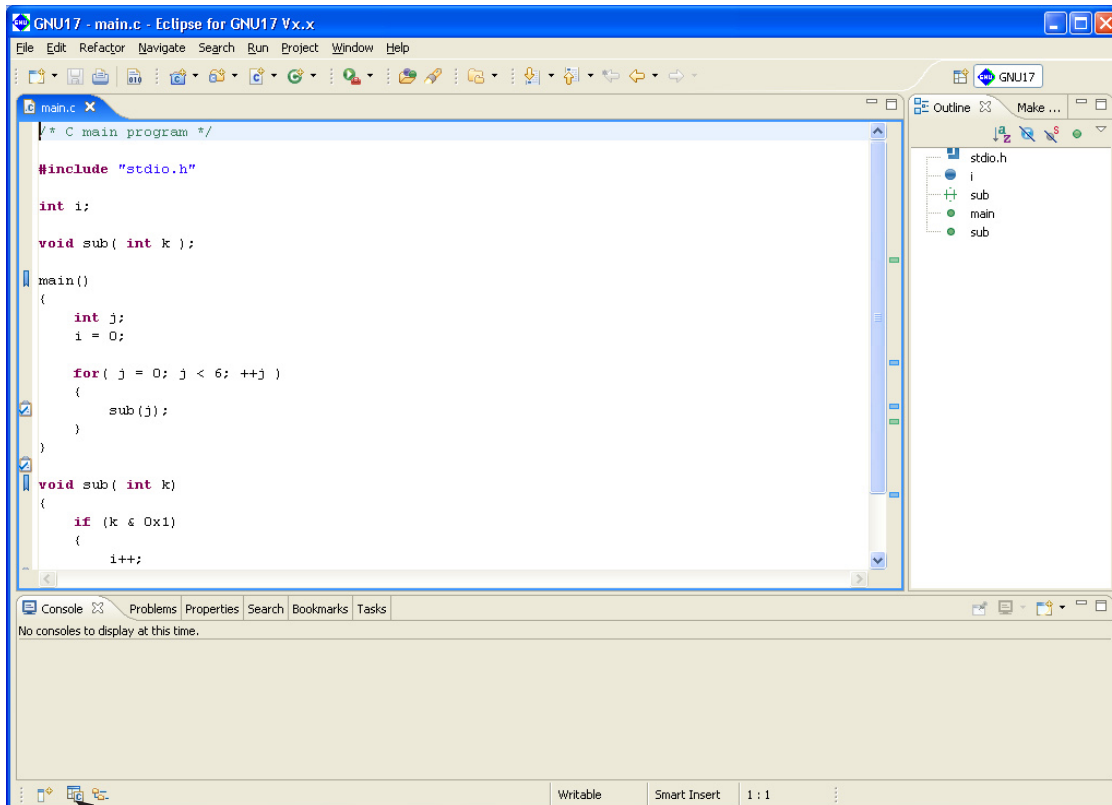
## Fast view

You may want to expand the editor area under certain circumstances — for example, when editing a source file. You can choose to maximize the editor area or use fast view mode instead. In this mode, views not currently required are temporarily iconized and the icon placed in a fast view area at the lower left corner of the window. You can click the icon to enlarge the view. Other views will not be hidden.



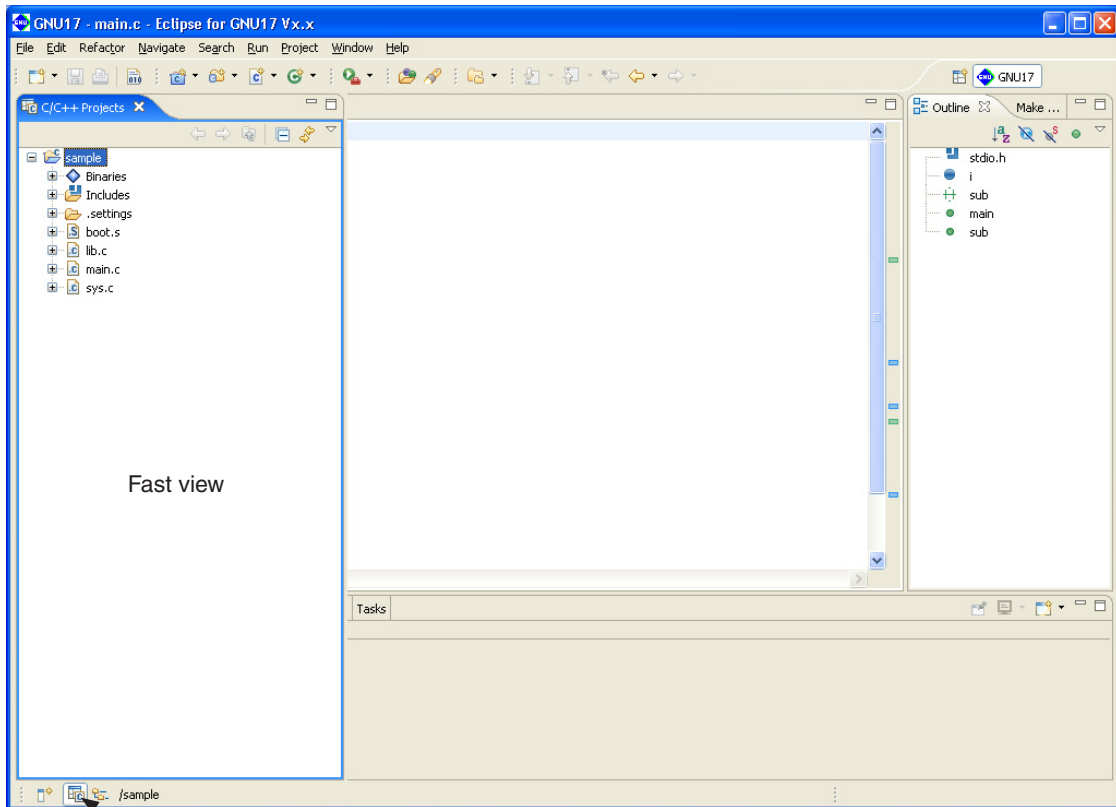
To turn a view into a fast view, right-click on the corresponding tab and select [Fast View] from the subsequent context menu. Or drag-and-drop the tab corresponding to a view into a fast view area to turn it into a fast view.

For example, select [Fast View] from the context menus of the [C/C++ Projects] and the [Navigator] tabs. The corresponding views will be turned into fast views, and icons will appear in the fast view area at the lower left corner of the window. The editor and lower view areas are enlarged as the views in the left pain are closed.

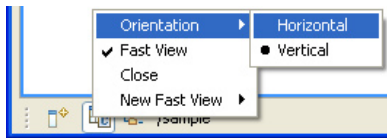


Fast view area

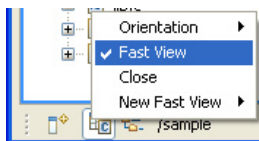
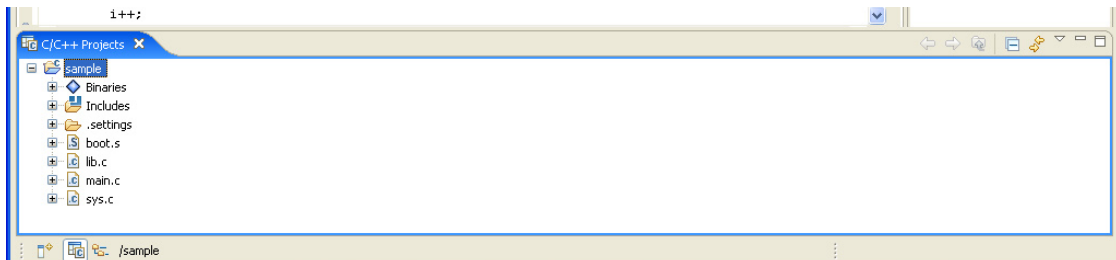
To open a fast view, click the corresponding icon in the fast view area. To close a fast view, click on any other view, or click the corresponding icon in the fast view area or the minimize button of the fast view.



Click the icon to open or close a fast view



By default, a fast view will be displayed vertically at the left edge of the window. For a different orientation, select [Orientation] > [Horizontal] from the context menu for the fast-view icon. The fast view will open horizontally at the bottom of the window.



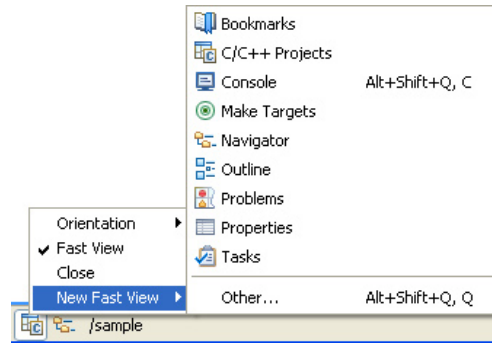
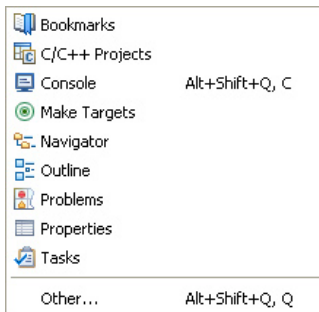
To restore a fast view to normal view, select [Fast View] from the context menu for the fast-view icon.

Select [Close] from this context menu to close the view and remove the icon.

In addition to the above, any view can be selected from the menu displayed by clicking the [New Fast View] button to open as a fast view. The same function can be performed from [New Fast View] in the context menu.



[New Fast View] button



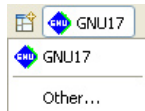
### To restore view layout to default settings

Select [Reset Perspective] from the [Window] menu. When a dialog box for confirmation is displayed, click [OK]. The view layout will revert to default settings of the **IDE** (the initial state when the **IDE** is started for the first time).

The view layout is saved when you quit the **IDE**. When you next start the **IDE**, it will start with the layout last saved. The **IDE** will not revert to the default settings when you restart it.

### 5.3.15 Perspectives

Perspectives represent the definitions of the configuration of displayed views, the view layout including the editor area, and the configuration of menus and toolbars. The version of Eclipse adopted for the **IDE** permits switching of perspectives to the these suiting particular development environments. In the **IDE**, the perspective named "GNU17" is defined with the view configuration and layout described in the preceding sections.



The word "GNU17" shown to the right of the perspective shortcut icon indicates that the GNU17 perspective is currently selected. Although the **IDE** allows you to switch perspectives, use only the default "GNU17" perspective.

## 5.4 Projects

### 5.4.1 What Is a Project?

The **IDE** manages individual applications being developed under a project name, creating a directory with the name you specified before beginning to develop an application, managing resources such as source files and files generated by the compiler and other tools in it.

In addition, project management files (`.cdtproject`, `.gnu17project`, and `.project`) are generated in a project directory and are updated from time to time by the **IDE**.

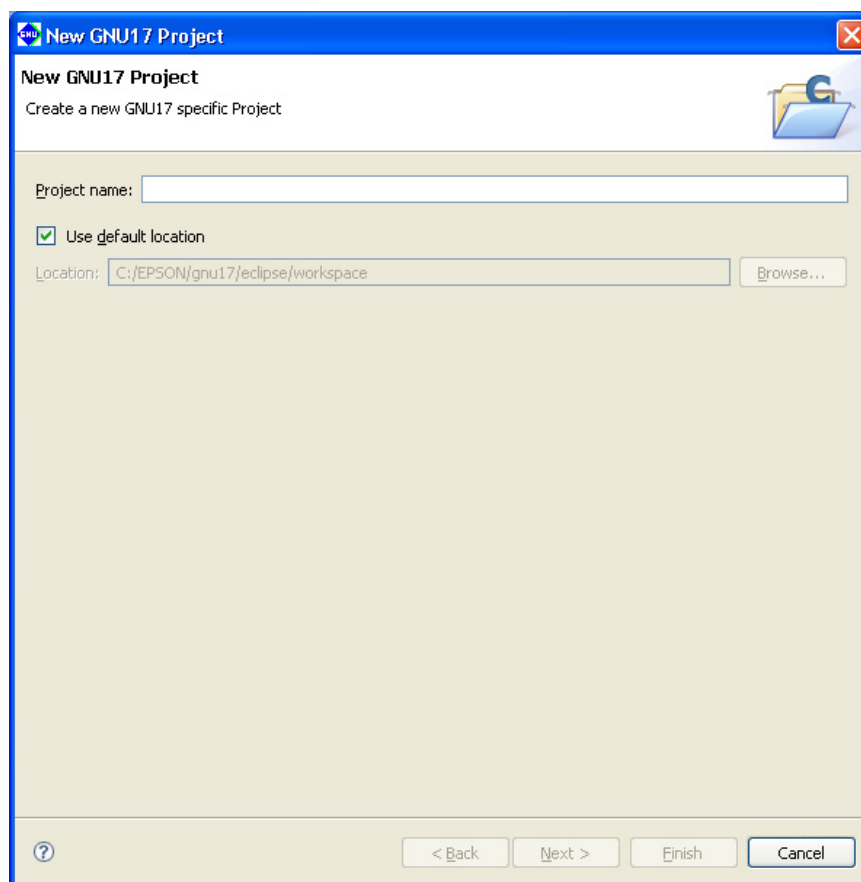
**Note:** These project management files which reside in the project directory must not be edited, moved, or deleted except when you manipulate them in the **IDE**. Attempting to do so will prevent you from restarting the project.

### 5.4.2 Creating a New Project

Application development by the **IDE** starts with creating a new project:

- (1) Launch the [New GNU17 Project] wizard by one of the following methods.
  - Select [New] > [New GNU17 Project] from the [File] menu.
  - Select [New GNU17 Project] from the [New] shortcut in the toolbar.
  - Select [New] > [New GNU17 Project] from the context menu for the [C/C++ Projects] or [Navigator] view.

The wizard will start, displaying the dialog box shown below.



- (2) Enter a project name in the [Project name:] text box.

**Notes:**

- Make sure the project name you enter is 100 characters or less.
- Only single-byte alphanumeric characters and underscores may be used for project names.

- (3) Specify the location at which you want to create a project directory. (This is necessary if you want to specify a specific location.)

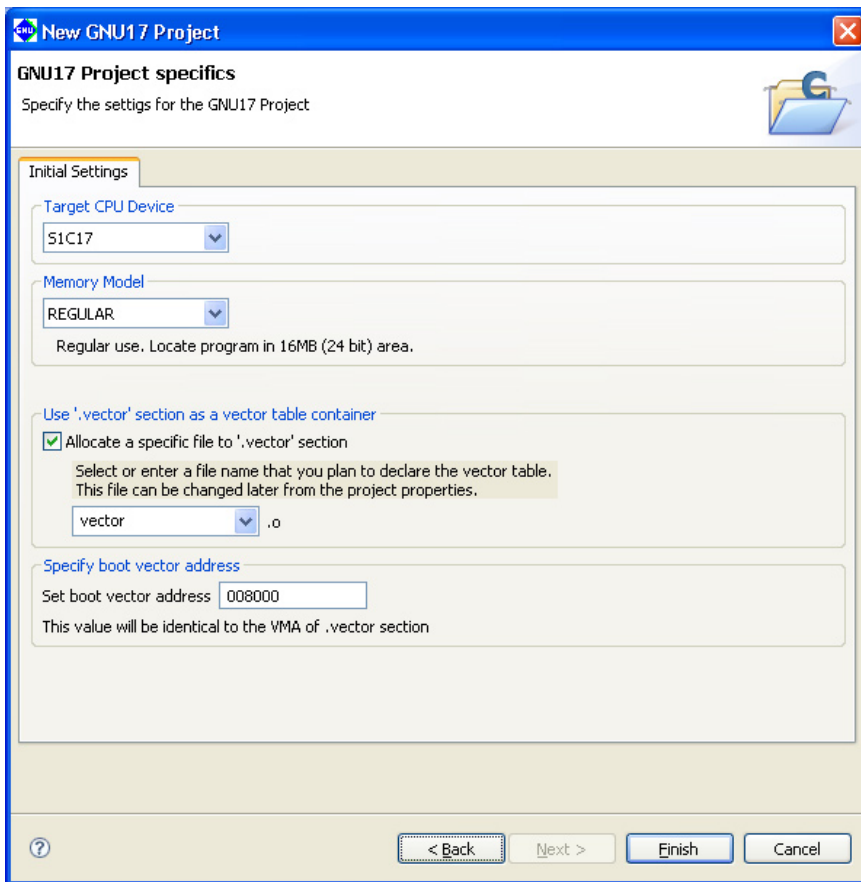
With default settings, the [Use default location] check box is selected, and a project directory is generated in the workspace directory specified when you started the **IDE**. Normally, go to the next step directly.

If you want to create a project directory outside the workspace, deselect the [Use default location] check box and enter a path in [Location:], or select an existing directory from the list displayed by clicking the [Browse...] button.

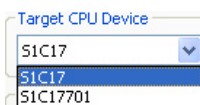
**Note:** The path is limited to a maximum of 200 characters.

- (4) Click the [Next>] button.

The **IDE** goes to the target CPU select screen shown below.



- (5) From the [Target CPU Device] combo box, select the target processor:

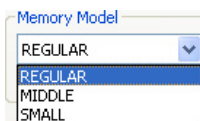


**S1C17:** Built-in S1C17 Core model  
**S1C17701:** When S1C17701 is the target model or when **ES-Sim17** is used for debugging

You can switch target CPUs later. (Refer to Section 5.7.1, "Setting the CPU Type and Memory Model".)

The models displayed in the list may be added/deleted by the configuration file that will be modified when a new model is released or an existing model is discontinued.

- (6) From the [Memory Model] combo box, select the memory model of the target:



**REGULAR:** 24 bits (Up to 16M-byte space can be used.)  
**MIDDLE:** 20 bits (Up to 1M-byte space can be used.)  
**SMALL:** 16 bits (Up to 64K-byte space can be used.)

You can modify settings for the memory model later. (Refer to Section 5.7.1, "Setting the CPU Type and Memory Model".)



- (7) Select an object you want to locate in the `.vector` section (section for a vector table) by selecting from the combo box (`vector.o` or `boot.o` selectable) or entering one in the box.



If no objects are to be located in the `.vector` section, deselect the check box entitled [Allocate a specific file to `.vector` section].

You can modify settings for the `.vector` section later. (Refer to Section 5.7.6, "Editing a Linker Script".)

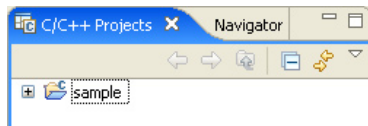
- (8) Specify a boot vector address. The default value is "008000". The value set here will be used as the parameter for the TTBR setting command that will be written in the debugger startup command file created by the **IDE** as well as it will be used as the VMA of the `.vector` section that will be written in the linker script file.
- (9) Click the [Finish] button.

The [New GNU17 Project] wizard is closed, and a project is created under the name specified.

Creating a new project creates a directory with the same name as the project in the current workspace or the directory specified in (3). If a directory with this project name already exists, the **IDE** uses it as the project directory.

In the [C/C++ Projects] or [Navigator] view, the project will be displayed along with a directory icon similar to the one shown below.

Example: Project created with the name "sample"

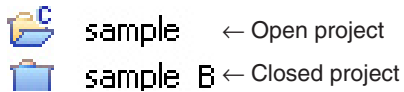


If multiple projects are created in the workspace, all will appear in the [C/C++ Projects] or [Navigator] view.

### 5.4.3 Opening and Closing a Project

When you create a project, the project stays in an open state. An open project remains open until you explicitly close it. Even when you restart the **IDE**, you can continue to work on that open project unless you have closed it.

Example: Icons of open and closed projects



In order to edit source files or to perform a build and other operations, the project must be open, and the project directory or contained files must be selected in the [C/C++ Projects] or [Navigator] view.

#### Closing a project

If you have more than one project in the workspace, you can close all of them except the one you are currently working on.

- (1) Select the project you want to close by clicking it in the [C/C++ Projects] or [Navigator] view.
- (2) Close the project by one of the following methods.
  - Select [Close Project] from the [Project] menu.
  - Select [Close Project] from the context menu for the [C/C++ Projects] or [Navigator] view.

This closes the project. At this time, any source files open in the editor are closed. If the contents edited in the editor have not been saved, the [Save Resources] dialog box (see Section 5.10.2) is displayed, letting you choose to save or not save the files file-by-file.

#### Opening a project

A project present within the workspace and closed in the [C/C++ Projects] or [Navigator] view can be opened by one of the following methods.

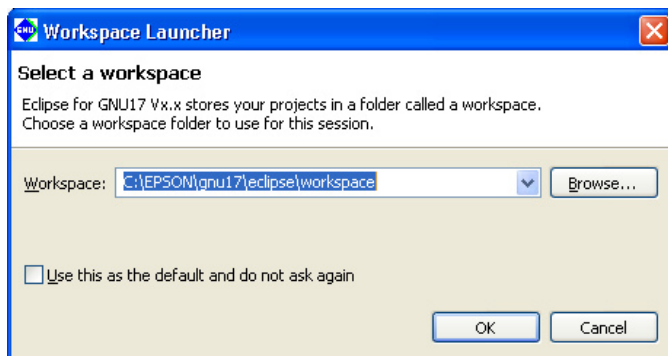
- (1) Select a project by clicking on it in the [C/C++ Projects] or [Navigator] view.
- (2) Open the selected project by one of the following methods.
  - Select [Open Project] from the [Project] menu.
  - Select [Open Project] from the context menu for the [C/C++ Projects] or [Navigator] view.

This works only for currently closed projects displayed in the [C/C++ Projects] or [Navigator] view. To open a project present in a directory outside the current workspace, switch the workspace to the directory in which the project is saved (see Section 5.4.4). Or import an existing project you want to open into the current workspace (see Section 5.4.5).

## 5.4.4 Switching Workspaces

The projects displayed in the [C/C++ Projects] or [Navigator] view are only those present in the current workspace. To perform any operation on a project in another directory, you must switch workspaces to that directory, or create a new directory and make it the workspace:

- (1) Save any documents currently being edited.
- (2) Select [Switch Workspace...] from the [File] menu.  
The [Workspace Launcher] dialog box is displayed.



- (3) Enter a path in the [Workspace:] combo box or select an existing directory from the directory select dialog box displayed by clicking the [Browse...] button.  
If the desired directory is a workspace previously used, select it from the list displayed by clicking the ▾ button in the [Workspace:] combo box.
- (4) Click the [OK] button.

The **IDE** window is temporarily closed. After the specified directory is set to the workspace, a new window appears. If the directory contains any open source files, these files are closed simultaneously with the **IDE** window. If the contents edited in the editor have not been saved, the [Save Resources] dialog box (see Section 5.10.2) is displayed.

If the workspace to which you've switched contains any existing projects, the window is opened in the status at the time you finished work on that project.

Specifying a nonexistent directory in (3) will create a new instance of your specified directory.

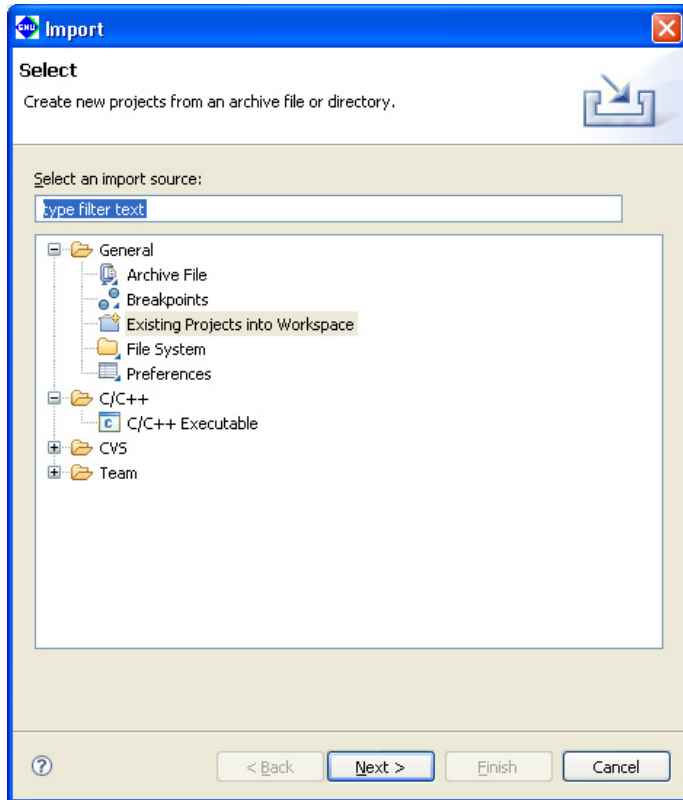
## 5.4.5 Importing an Existing Project

This section describes how to import an existing project into the current workspace. The import procedure is described below.

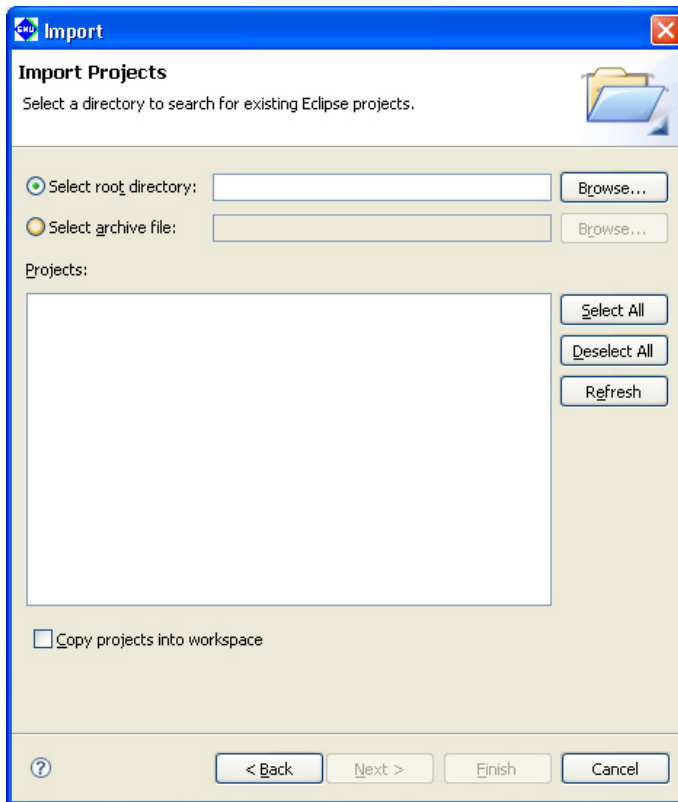
(1) Perform one of the operations described below.

- Select [Import...] from the [File] menu.
- Select [Import...] from the context menu for the [C/C++ Projects] or [Navigator] view.

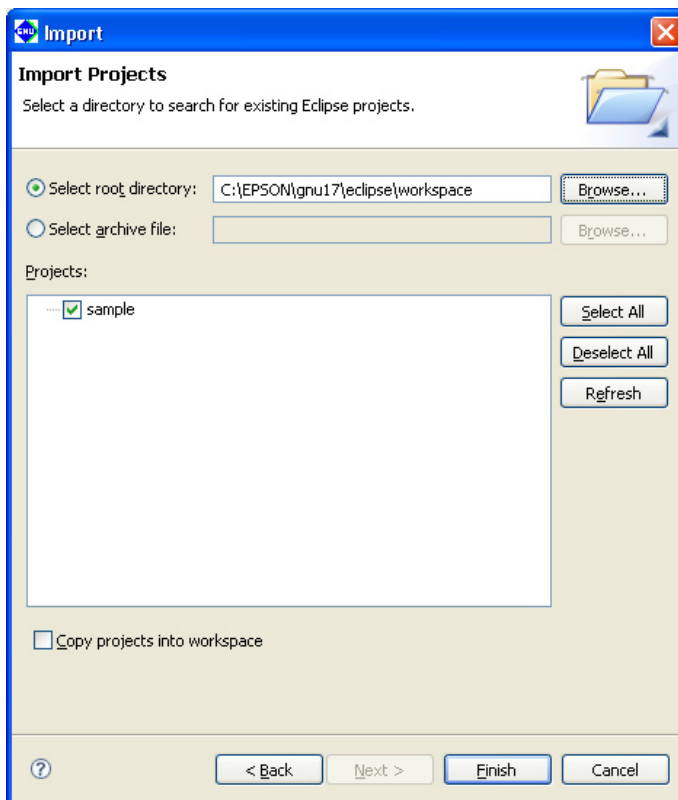
The [Import] wizard will start.



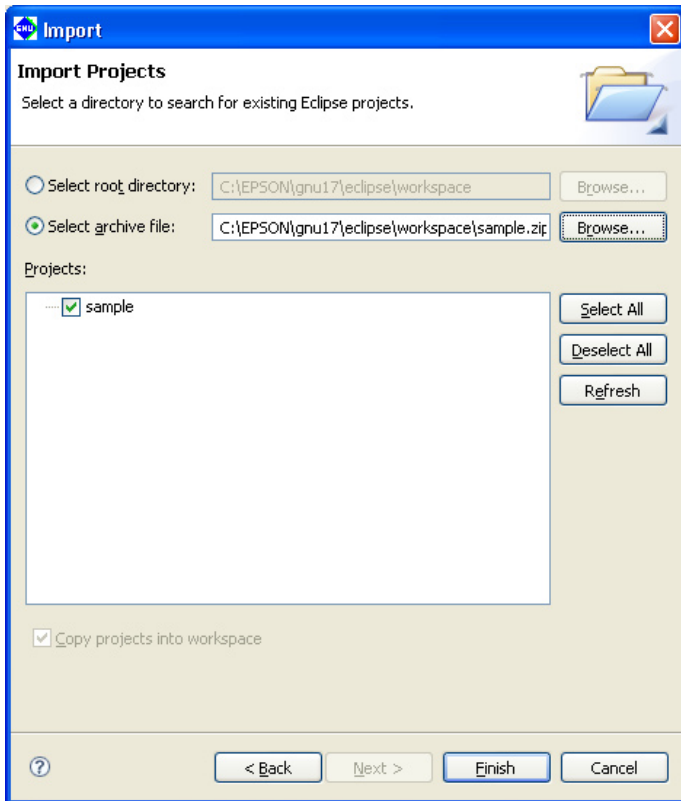
(2) Select [Existing Projects into Workspace] from the list and click [Next>].



- (3) When the project is not archived, select the [Select root directory:] radio button. Then select the project directory you want to import in the directory select dialog box displayed by clicking the [Browse...] button.



When the project is an archived file, select the [Select archive file:] radio button. Then select the project archived file in the file select dialog box displayed by clicking the [Browse...] button.



When the root directory or the archived file has been selected, the projects that exist in it are displayed in the [Projects:] list box. Select the check box for the project to be imported (one or more projects can be selected). The [Select All] button is used to select all the projects in the list and the [Deselect All] button is used to deselect all the project in the list.

The [Refresh] button brings the list up to date.

The [Copy projects into workspace] check box is used to select whether the project is copied into the workspace directory or not.

#### When not copying the project

Deselect the [Copy projects into workspace] check box. The project will not be copied into the workspace and editing operations will be applied to the files located in the original project directory. Be aware that the original project folder is deleted by the operation to delete the project.

#### When copying the project

Select the [Copy projects into workspace] check box. The specified project directory will be copied into the workspace and editing operations will be applied to the files located in the workspace. The files located in the original project directory are left unmodified.

Be sure to select the [Copy projects into workspace] check box if you do not want to change the original files.

When an archived file is selected, the projects in it are always copied into the workspace.

(4) Click the [Finish] button.

The imported project will be displayed in the [C/C++ Projects] or [Navigator] view.

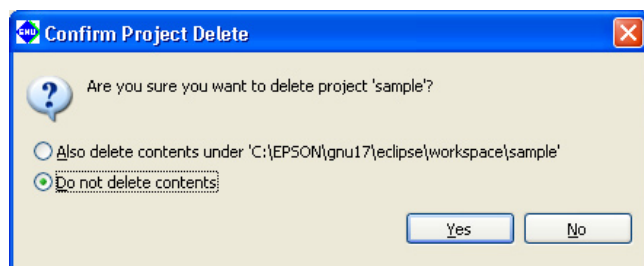
You also can import and execute a build process on a project created in another environment (PC) into the current PC by copying it in its entirety, including the project directory. However, if the project is configured with exclusive include search paths and library paths in the build options, you may have to modify these paths after importing the project.

## 5.4.6 Deleting a Project

Unnecessary projects can be deleted as described below.

- (1) Select a project you want to delete by clicking on it in the [C/C++ Projects] or [Navigator] view.
- (2) Perform one of the operations described below.
  - Select [Delete] from the [Edit] menu.
  - Select [Delete] from the context menu for the [C/C++ Projects] or [Navigator] view.
  - Press the [Delete] key.

The [Confirm Project Delete] dialog box is displayed.



\* To leave the directories and files in the file system intact

- (3) Select the [Do not delete contents] radio button.

In this case, although the project displayed in the [C/C++ Projects] or [Navigator] view disappears, the files remain intact. If you import the same project (see Section 5.4.5), you can continue working on it as a project again.

\* To delete all directories and files in the file system along with the project

- (3') Select the [Also delete contents under '<path>'] radio button.

**Note:** Keep in mind that if you select this option, all files associated with the project are deleted from the disk, and the project can no longer be recovered.

- (4) Click the [Yes] button. To cancel, click the [No] button.

**Note:** If you created a project with a project name exceeding the permissible maximum number of characters, the project directory may not be deleted. In such cases, quit the **IDE** and rename the directory name of the project from the shell (i.e., the command prompt) before deleting the project.

## 5.4.7 Changing the Project Name

To change a project name in the [C/C++ Projects] or [Navigator] view, follow the procedure described below.

- (1) Select a project whose name you want to change by clicking on it in the [C/C++ Projects] or [Navigator] view.
- (2) Perform one of the following operations.
  - Select [Rename...] from the [File] menu.
  - Select [Rename] from the context menu for the [C/C++ Projects] or [Navigator] view.
- (3) The project name in the view will be placed in editing mode. Enter a new name and press the [Enter] key.

This operation is reflected in the file system.

Changing a project name also changes the project directory name as well as the following:

- Command file name (*<project name>\_gnu17IDE.cmd*)  
The previous file *<old project name>\_gnu17IDE.cmd* file is not deleted.
- The names of the files in the project listed below (changed the next time you build)
  - Executable format object file (*<project name>.elf*)
  - Makefile (*<project name>\_gnu17IDE.mak*)\*
  - Linker script file (*<project name>\_gnu17IDE.lds*)\*
  - Parameter file (*<project name>\_gnu17IDE.par*)\*

\* These files are deleted when you change a project name, and are newly generated the next time you build.  
The elf file already generated is not deleted and remains intact with its previous name.
- Debugger startup settings  
The contents set in the [External Tools] dialog box are changed according to a new project name.

- Notes:**
- Only single-byte alphanumeric characters and underscores can be used in a project name. Keep in mind that including any other characters or symbols in a project name will result in an error when you perform a build or other operation.
  - When the project name is changed, the **IDE** generates a new command file using the template without copying the contents of the previous command file. Therefore, copy the contents of the *<old project name>\_gnu17IDE.cmd* and paste them in to the [Properties] > [GNU17 GDB Commands] page as necessary.



## 5.4.8 Resource Manipulation in a Project

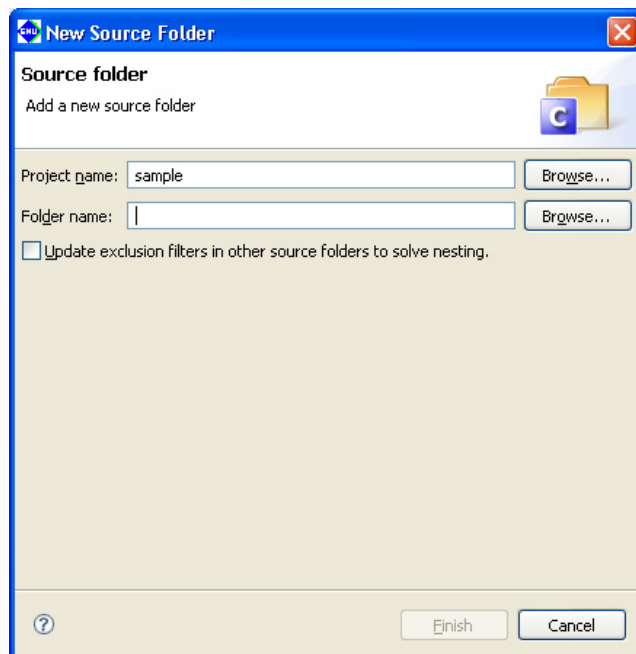
This section describes the operations to create resources, as well as importing, copying, moving, or deleting resources that are possible in the [C/C++ Projects] or [Navigator] view.

### Creating a new source directory

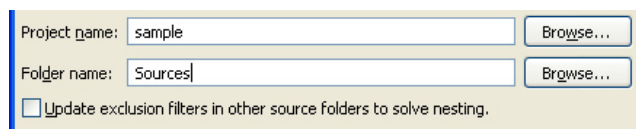
You can create a new source directory in a project:

- (1) Perform one of the following operations.
  - Select [New] > [Source Folder] from the [File] menu.
  - Select [New] > [Source Folder] from the context menu for the [C/C++ Projects] or [Navigator] view.
  - Select [Source Folder] from the [New] shortcut in the toolbar.
  - Click the [New C/C++ Source Folder] button in the toolbar.

The [New Source Folder] dialog box is displayed.



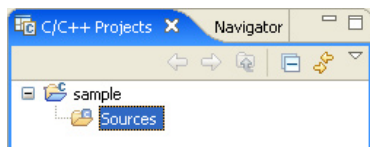
- (2) The current project name is entered in [Project name:]. When creating a source directory in another project, select the project directory using the [Browse...] button.
- (3) Enter the name of the directory you want to create in the [Folder name:] text box.



If source files are already imported into the project, select the [Update exclusion filters in other source folders to solve nesting.] check box. When a source file is created without selecting the check box, the source files must be imported again.

- (4) Click the [Finish] button. To cancel, click the [Cancel] button.

The directory you created is displayed.



Although a general directory is created on the file system, it is assumed as a source directory. The source files located in the source directory will be included to the make process.

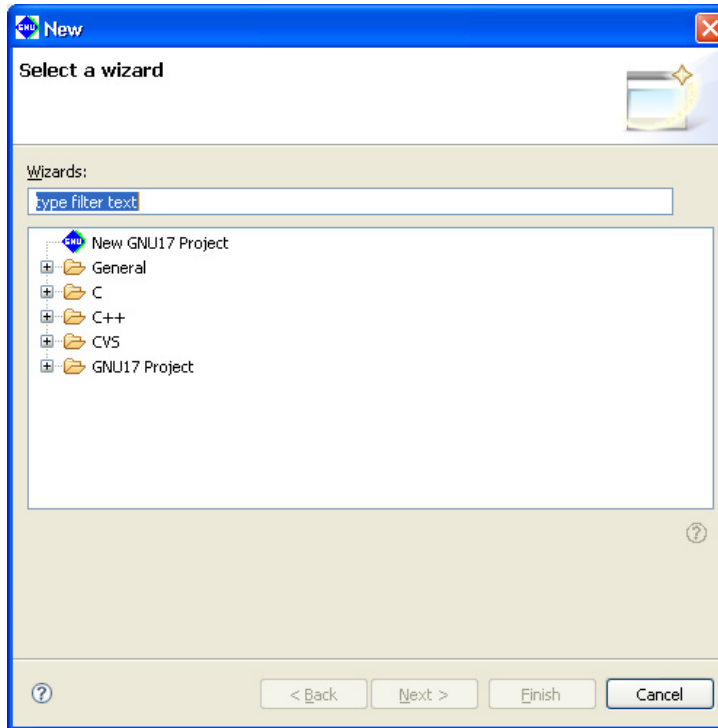
## Creating a new directory for general-purpose use

You can create a new directory in a project or in the internal directory of a project:

(1) Perform one of the following operations.

- Select [New] > [Other...] from the [File] menu.
- Select [New] > [Other...] from the context menu for the [C/C++ Projects] or [Navigator] view.
- Select [Other...] from the [New] shortcut in the toolbar.

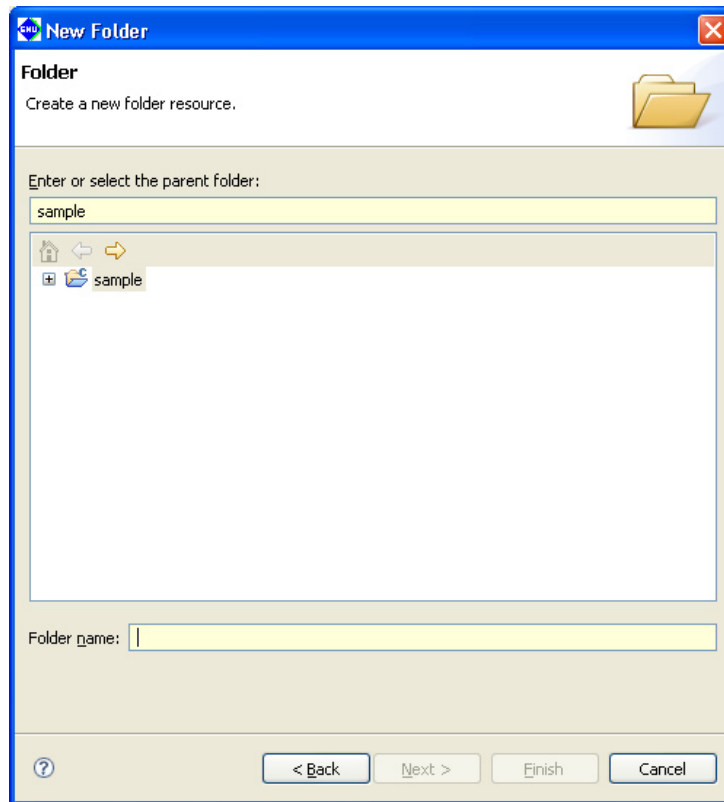
The [New] dialog box is displayed.



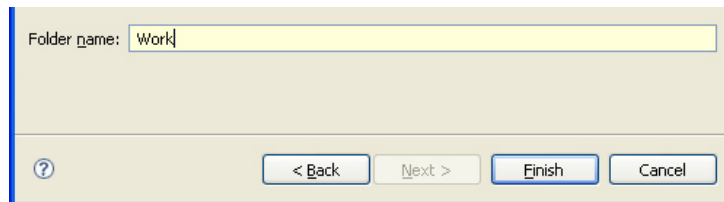
(2) Select [Folder] from the [General] tree list.



(3) Click [Next >].

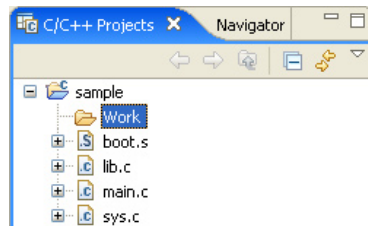


- (4) Select the project or the parent directory in which you want to create a new directory (subdirectory) by clicking on it in the directory list in tree form.
- (5) Enter the name of the directory you want to create in the [Folder name:] text box.



- (6) Click the [Finish] button. To cancel, click the [Cancel] button.

The directory you created is displayed.



**Note:** The source files located in this folder will not be included in a make process. Place the source files into a source folder to include them in a make process.

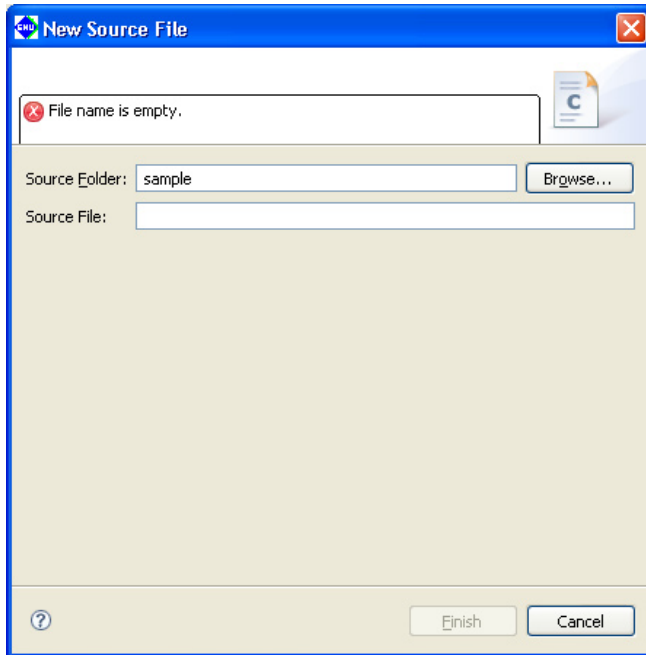
## Creating a new source file/header file

You can create a new source or header file in a project:

(1) Perform one of the following operations.

- Select [New] > [Source File] (to create a source file) or [Header File] (to create a header file) from the [File] menu.
- Select [New] > [Source File] or [Header File] from the context menu for the [C/C++ Projects] or [Navigator] view.
- Select [Source File] or [Header File] from the [New] shortcut in the toolbar.
- Select [Source File] or [Header File] from the [New C/C++ Source File] shortcut in the toolbar.
- Click the [New C/C++ Source File] button in the toolbar (to create a source file).

The [New Source File] (or [New Header File]) dialog box is displayed.

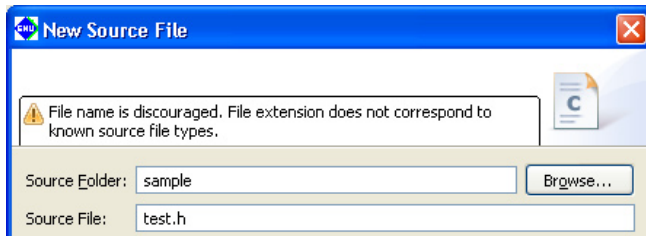


(2) Enter the name of the file you want to create in the [Source File:] (or [Header File:]) text box.



Enter the file extension as ".c" to create a C source file or ".s" to create an assembler source file.

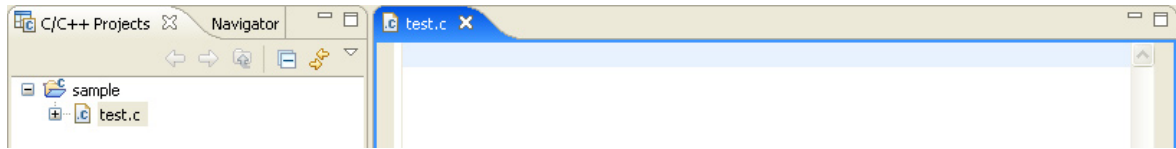
The message shown below appears if an appropriate file extension for the source file is not entered. However, the file can be created with the entered name even if another file extension is specified.



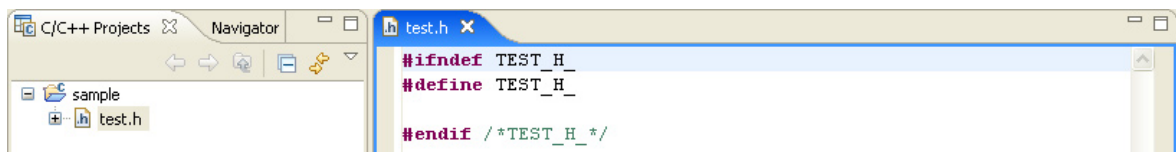
The current project name is entered in [Source Folder:]. When creating the file in another directory, enter the path or select the directory using the [Browse...] button.

(3) Click the [Finish] button. To cancel, click the [Cancel] button.

The file you created is displayed in the view and open with the editor.



When a header file is created by selecting [Header File] from a menu, the created file contains a macro definition (<file name>\_H\_) described automatically.

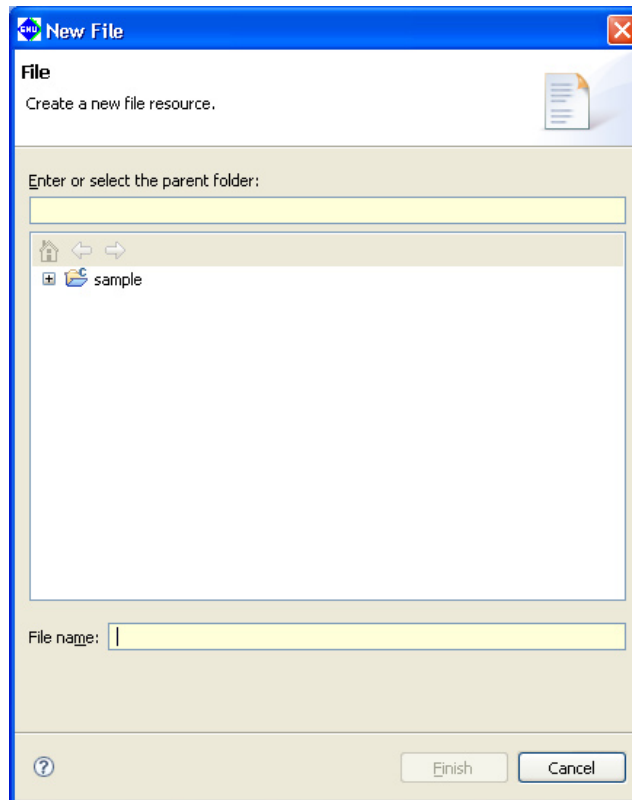


### Creating a new general-purpose text file

You can create a new text file in a project or in the internal directory of a project:

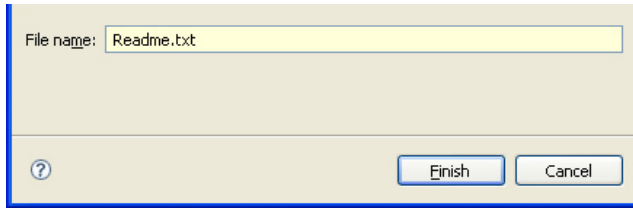
- (1) Perform one of the following operations.
  - Select [New] > [File] from the [File] menu.
  - Select [New] > [File] from the context menu for the [C/C++ Projects] or [Navigator] view.
  - Select [File] from the [New] shortcut in the toolbar.
  - Select [File] from the [New C/C++ Source File] shortcut in the toolbar.

The [New File] dialog box is displayed.



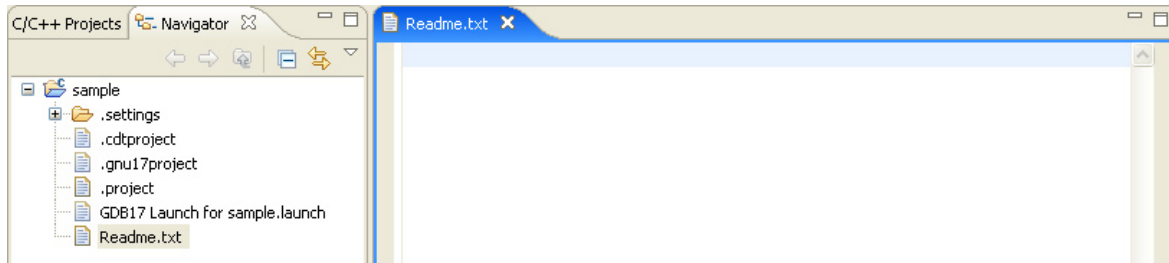
- (2) Select the project or the parent directory in which you want to create a new file by clicking on it in the directory list in tree form.

- (3) Enter the name of the file you want to create in the [File name:] text box.



- (4) Click the [Finish] button. To cancel, click the [Cancel] button.

The file you created is displayed in the view\* and open with the editor.



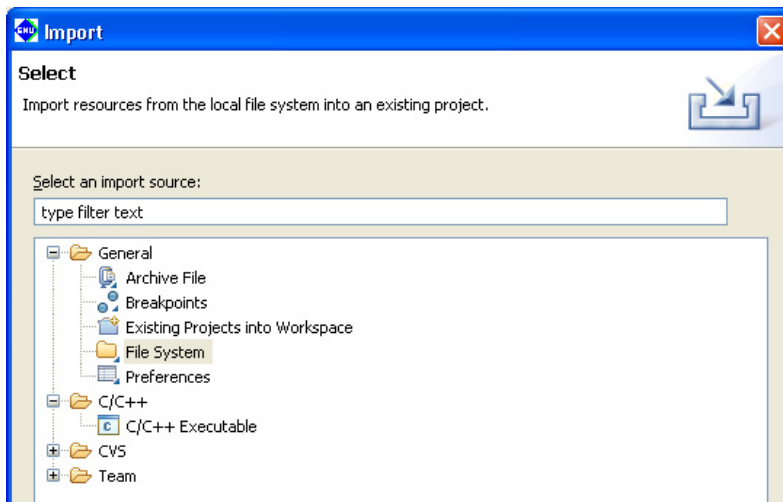
\* The file is not displayed in the [C/C++ Projects]/[Navigator] view when the file type (extension) is disabled to display by the filter setting.

## Importing an existing file or directory

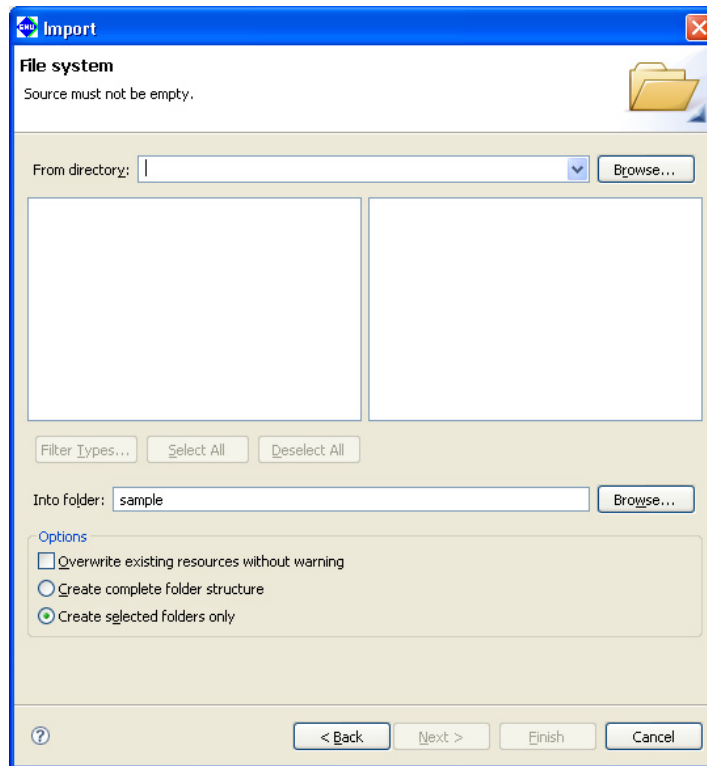
Described below is the procedure for importing an existing file or directory into a project. In the import procedure described here, files/directories are also copied to a project directory in the file system. This approach allows you to retain the original unchanged files when (for example) creating a new source by correcting an existing source. The import procedure is given below.

- (1) From within the [C/C++ Projects] or [Navigator] view, click on the project or directory into which you want to import a file/directory.
- (2) Do one of the following:
  - Select [Import...] from the [File] menu.
  - Select [Import...] from the context menu in the [C/C++ Projects] or [Navigator] view.

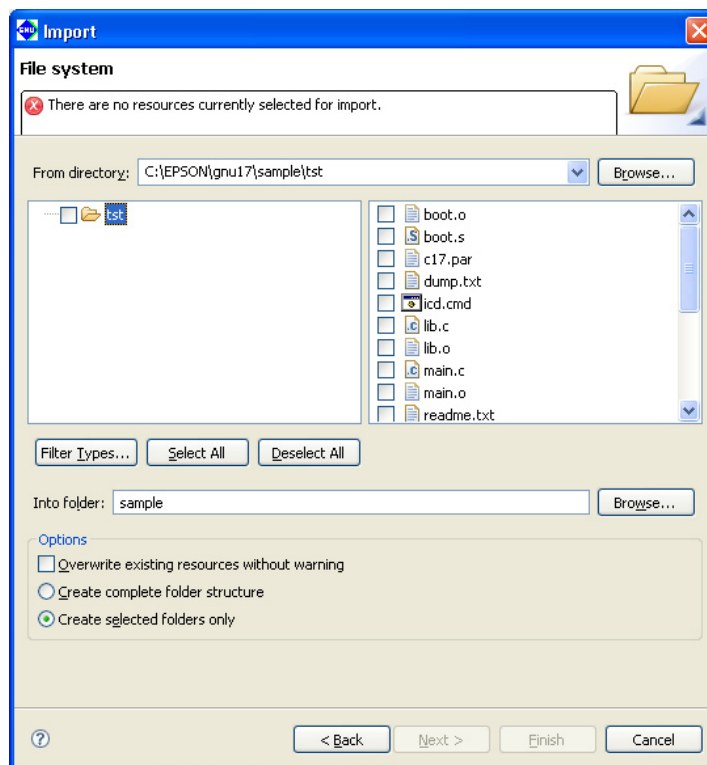
This launches the [Import] wizard.



- (3) Select [File System] from the list and click [Next>].



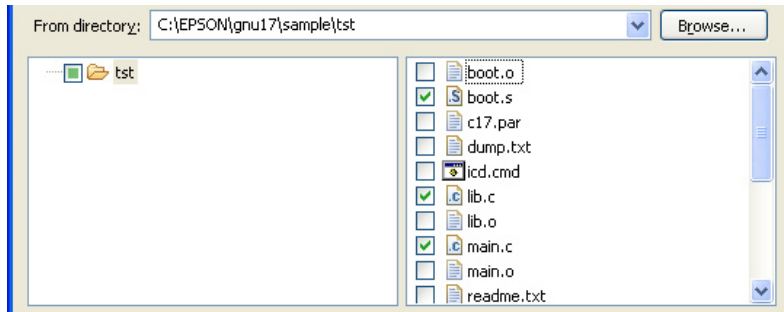
- (4) In the directory select dialog box displayed when you click the [Browse...] button to the right of the [From directory:] combo box, select the directory containing the file/directory you want to import. This populates the [From directory:] combo box with the path to the selected directory. If you imported from this project previously, you can select the project from the history displayed by clicking the ▾ button in the [From directory:] combo box.



**Note:** Always be sure to select the parent directory of the file/directory you want to import. The files/directories to be imported are located in the directory displayed as the root directory in the directory list to the left of the dialog box.

- (5) To import a file, select the file you want to import from the list box to the right (indicated by a check mark when selected).

Example: Importing a file



To import a directory, expand the tree list in the list box to the left and select the directory you want to import from the list of sub-directories. You can also select and import a sub-sub-directory, in which case the directory structure from the sub-directory is also imported (for files, only those in the selected directory are imported).

Example: Importing a directory



- (6) Click the [Finish] button.

The imported file/directory is displayed in the [C/C++ Projects] or [Navigator] view.

Refer to Section 5.10.3 for information on the [Import > File system] dialog box and other controls.

## Specifying a source folder

The **IDE** writes the source files located directly below the project directory and the source files located in the source folder of the project to a makefile, thereby specifying those files as targets to be assembled/compiled. In their initial default status, source files located in the internal directory of a project other than the source folder are not written in a makefile. However, you can register that directory as a source folder in project properties so that the source files in the directory are recognized as targets for the make command.

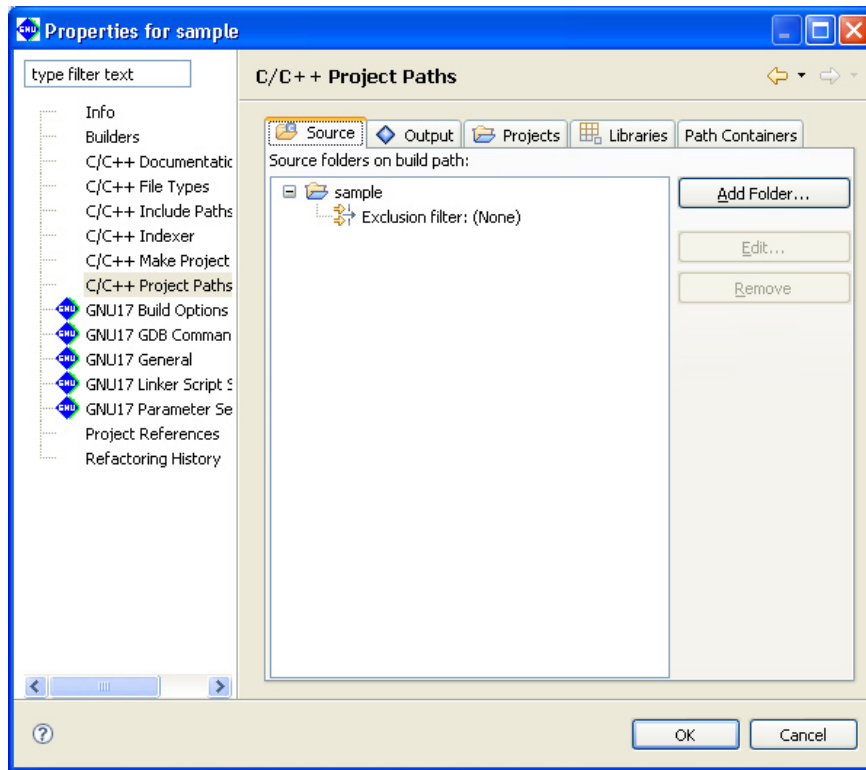
This procedure is described below. Note that the directory to be registered is assumed to have been already created in or imported into the project.

- (1) In the [C/C++ Projects] or [Navigator] view, select the project containing the source folder you want to select.
- (2) Do one of the following:
  - Select [Properties] from the [Project] menu.
  - Select [Properties] from the context menu in the [C/C++ Projects] or [Navigator] view.

This will display the [Properties] dialog box.

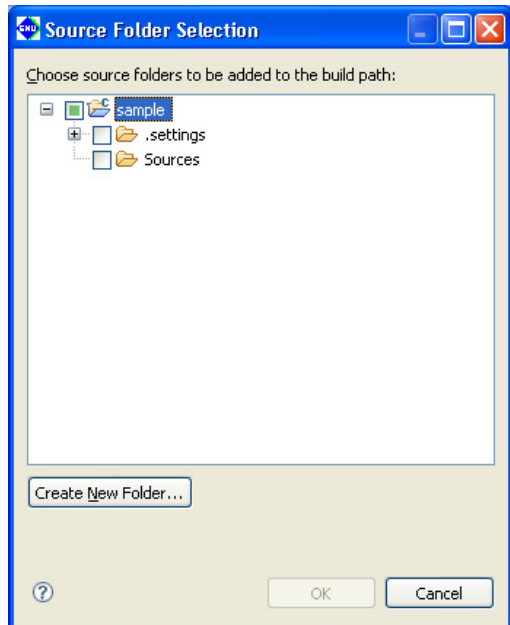
- (3) Select [C/C++ Project Paths] from the properties list, then click the [Source] tab.



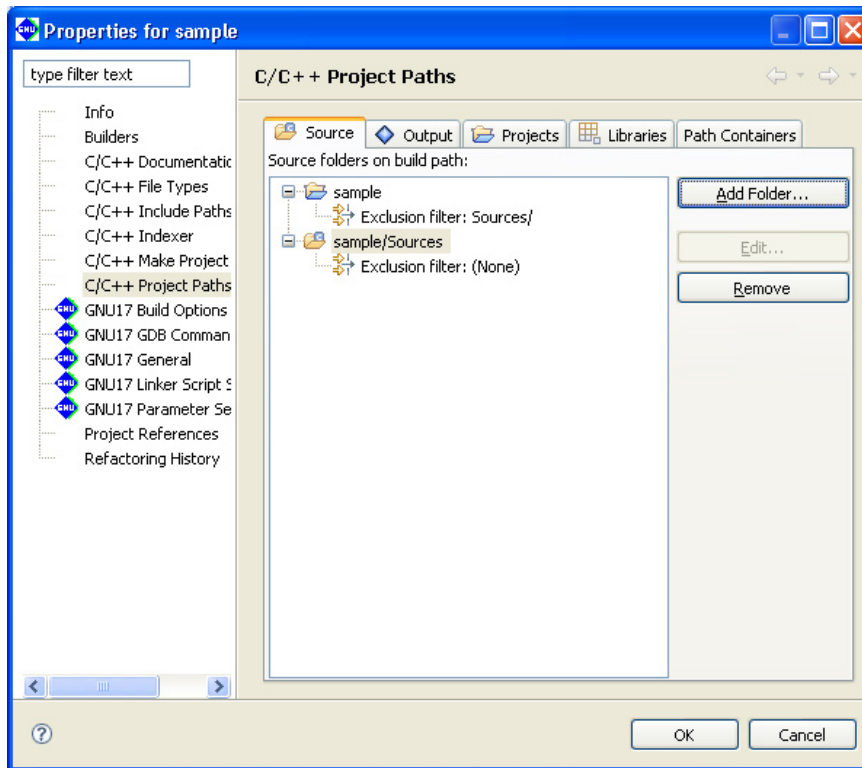


By default, project directories are already registered.

- (4) Click the [Add Folder...] button.  
This displays the [Source Folder Selection] dialog box.



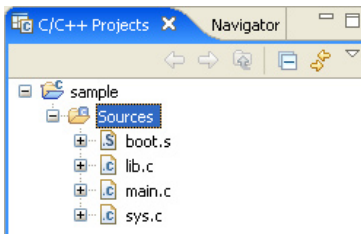
- (5) Select the directory you want to register (indicated by a check mark when selected) and click the [OK] button.



The selected directory will be registered in the list of source directories. To remove a registered source folder, select it from the list and click the [Remove] button. Use the [Edit...] button (select "Exclusion filter" from the list before use) to select any files in the source folder you want to exclude from a makefile.

(6) Click the [OK] button to close the [Properties] dialog box.

A source folder in the [C/C++ Projects] or [Navigator] view is indicated by the letter 'C' superimposed on its icon, as shown below.



## Copying/pasting a file or directory

You can copy and paste resources from within the [C/C++ Projects] or [Navigator] view.

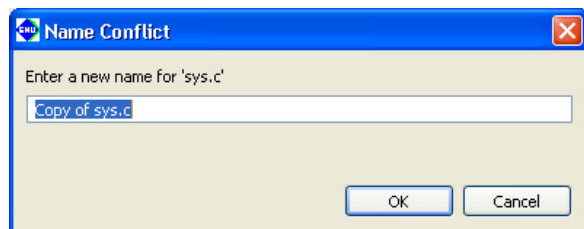
### Copying

- (1) Select the file or directory you want to copy in the [C/C++ Projects] or [Navigator] view.
- (2) Do one of the following:
  - Select [Copy] from the [Edit] menu.
  - Select [Copy] from the context menu in the [C/C++ Projects] or [Navigator] view.

This copies the selected resource to the clipboard.

## Pasting

- (1) In the [C/C++ Projects] or [Navigator] view, select the project or directory into which you want to paste the copied file or directory.
- (2) Do one of the following:
  - Select [Paste] from the [Edit] menu.
  - Select [Paste] from the context menu in the [C/C++ Projects] or [Navigator] view.
- (3) The dialog box shown below is displayed if you attempt to paste the file or directory to the location from which you copied it. Rename the file or directory, if necessary.



The copy-and-paste operations performed in the [C/C++ Projects] or [Navigator] view are also reflected in the file system.

You can paste resources copied in the [C/C++ Projects] or [Navigator] view into Windows Explorer or paste resources copied in Windows Explorer into the [C/C++ Projects] or [Navigator] view.

## Moving a file or directory

Do the following to move files or directories in the [C/C++ Projects] or [Navigator] view:

- (1) In the [C/C++ Projects] or [Navigator] view, select the file or directory you want to move.
- (2) Do one of the following:
  - Select [Move...] from the [File] menu.
  - Select [Move...] from the context menu in the [C/C++ Projects] or [Navigator] view.
- (3) A directory select dialog box is displayed. Select the directory into which you want to move the file or directory and click [OK].

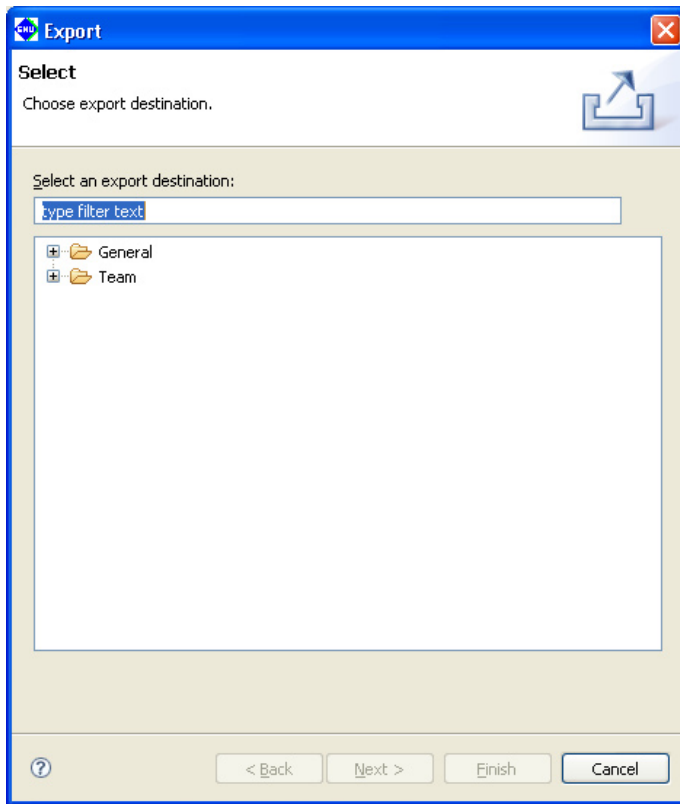
Move operations performed in the [C/C++ Projects] or [Navigator] view are reflected in the file system. This method is restricted to movements within the current workspace. To move a file or directory to a location outside the workspace, you must copy the resource by copying and pasting or exporting, then delete the resource from within the view.

## Exporting a file or directory

As described below, files/directories in the [C/C++ Projects] or [Navigator] view can be written out (exported) to outside the workspace.

- (1) In the [C/C++ Projects] or [Navigator] view, select the file or directory you want to export to the outside location.
- (2) Do one of the following:
  - Select [Export...] from the [File] menu.
  - Select [Export...] from the context menu in the [C/C++ Projects] or [Navigator] view.

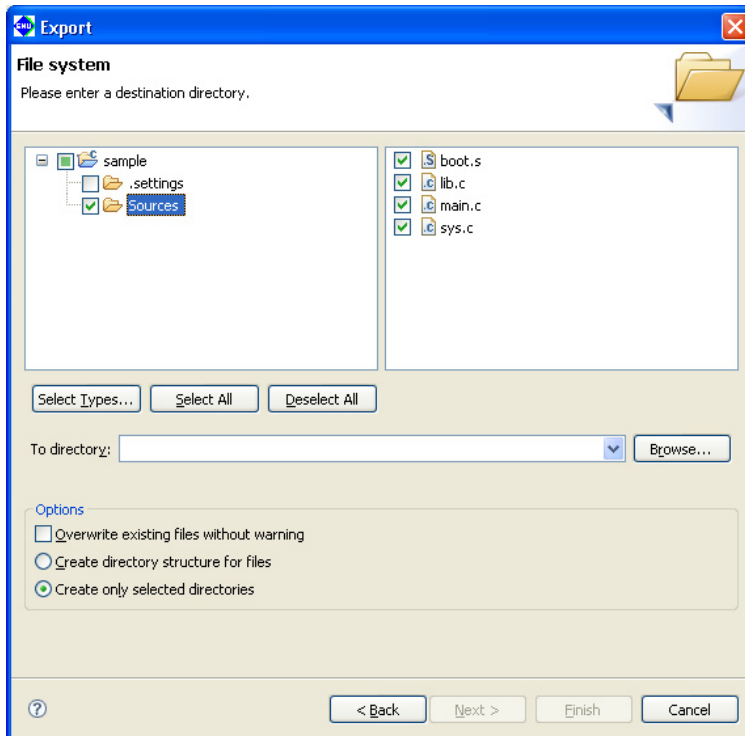
This launches the [Export] wizard.




(3) Expand [General].



(4) Select [File System] from the wizard list and click [Next>].



- (5) In the directory select dialog box displayed when you click the [Browse...] button to the right of the [To directory:] combo box, select the directory to which you want to export the file or directory. The [To directory:] combo box is populated by the path corresponding to the selected directory. If the directory is one to which you previously exported, you can select it from the history displayed when you click the  button in the [To directory:] combo box.
- (6) Select or deselect the check boxes in the directory list and file list to edit the directories or files you want to export.
- (7) Click the [Finish] button.

The selected directory/files are written out to the specified directory.

Refer to Section 5.10.4 for information on the [Export > File system] dialog box and other controls.

### Renaming a file or directory

Do the following to rename a file or directory in the [C/C++ Projects] or [Navigator] view:

- (1) In the [C/C++ Projects] or [Navigator] view, select the file or directory you want to rename.
- (2) Do one of the following:
  - Select [Rename...] from the [File] menu.
  - Select [Rename] from the context menu in the [C/C++ Projects] or [Navigator] view.
- (3) The file/directory name in the view will be placed in edit mode. Enter a new name and press the [Enter] key.

This operation is reflected in the file system.

### Deleting a file or directory

Do the following to delete a file/directory in the [C/C++ Projects] or [Navigator] view:

**Note:** Be careful when deleting a file/directory from the [C/C++ Projects] or [Navigator] view, since doing so will also delete the actual file/directory from the file system.

- (1) In the [C/C++ Projects] or [Navigator] view, select the file or directory you want to delete.
- (2) Do one of the following:
  - Press the [Delete] key.
  - Select [Delete] from the [Edit] menu.
  - Select [Delete] from the context menu in the [C/C++ Projects] or [Navigator] view.
- (3) A confirmation dialog box is displayed. Click [Yes] to delete or [No] to cancel.

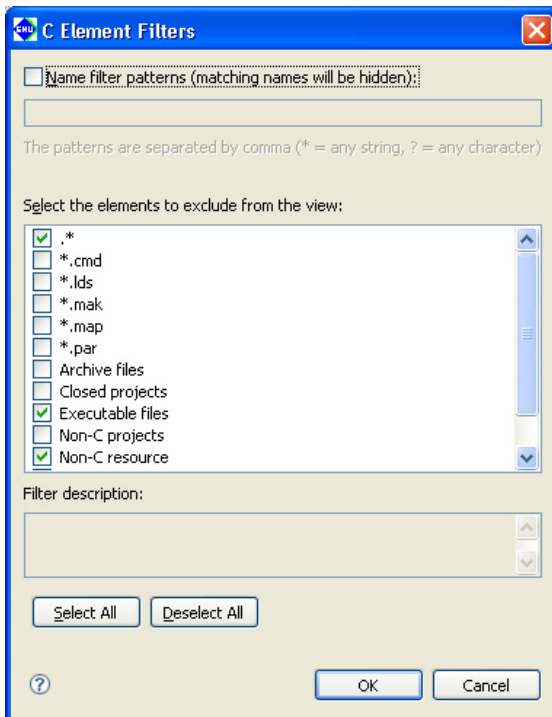
## 5.4.9 File Filter

File management and navigation in the **IDE** are performed in the [C/C++ Projects] or [Navigator] view. These views have a file filter function for screening out certain file types.

The file filter affects only the display in this view. Even if you choose not to display some of the source files needed to build a project, these files will be included in the build process, and the project build will proceed correctly.

### File filter in the [C/C++ Projects] view

Select [Filters...] from the toolbar menu (▽) in the [C/C++ Projects] view to display the dialog box shown below.



Select the type of file you want filtered out of the display. Use [Select All] to select all file types or [Deselect All] to deselect all file types.

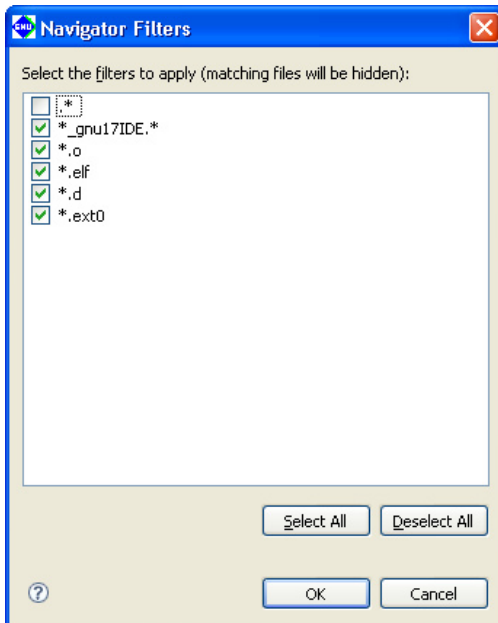
Select the [Name filter patterns:] check box and enter a character string in the text box. That allows you to filter out files whose names match the character string. You can use \* (string) and ? (one character) wildcards.

The files listed below are not displayed in the initial settings for the [C/C++ Projects] view:

- Files other than below corresponding to ".\*"
  - Executable files (.elf files)
  - Object files (.o files)
  - Text files not associated with C

### File filter in the [Navigator] view

Select [Filters...] from the toolbar menu (▽) in the [Navigator] view to display the dialog box shown below.



Select the type of file to filter from the display.

Use [Select All] to select all file types or [Deselect All] to deselect all file types.

By default, the files listed below are not displayed in the [Navigator] view:

- Files corresponding to "\*.o" (object files)
- Files corresponding to "\*.elf" (executable files)
- Files corresponding to "\*.d" (build dependency files)
- Files corresponding to "\*\_gnu17IDE.\*" (**IDE** files)
- Files corresponding to "\*.ext0" (assembler source files output from the compiler)

## 5.4.10 Working Set

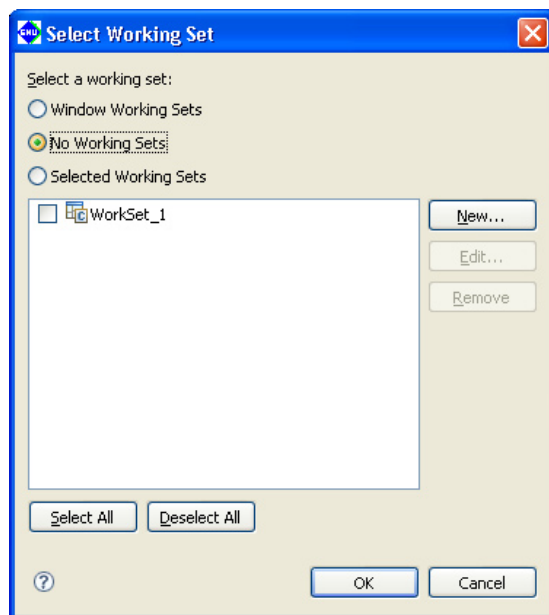
It is possible to group only the necessary resources as a working set and to limit the contents displayed in the [C/C++ Projects] or [Navigator] view to a specified working set. This section describes how to manipulate a working set in the [C/C++ Projects] view. The procedure described below is the same as for [Navigator] view.

The working set created as described below can also be used to specify a search domain or a project to build during a build process.

### Selecting a working set

**Note:** Working sets must be created in advance.

- (1) Select [Select Working Set...] from the toolbar menu (∇) in the [C/C++ Projects] view. This displays the [Select Working Set] dialog box, which shows a list of the working sets created.



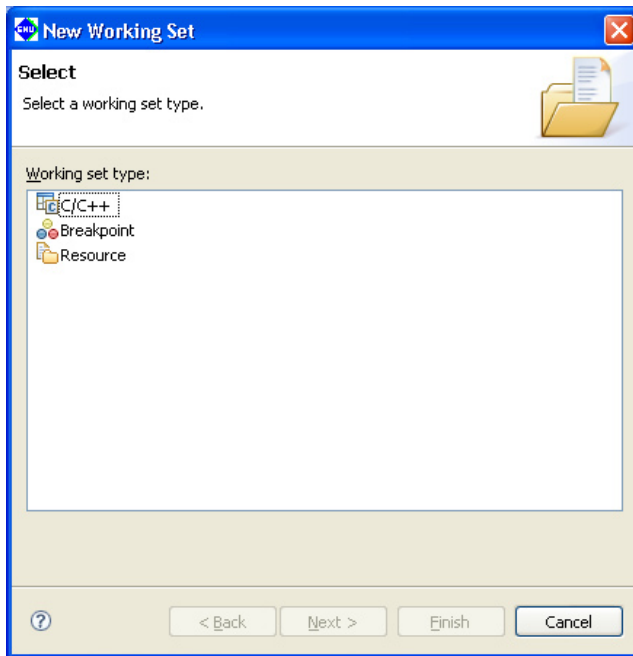
- (2) Select a working set from the list and click [OK].

Only the resources defined in the selected working set will be displayed in the [C/C++ Projects] view.

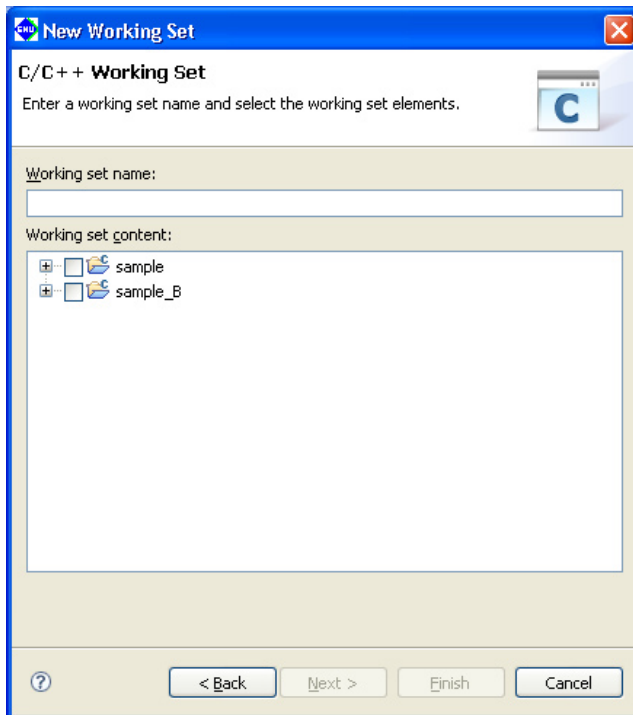
To revert the display in the [C/C++ Projects] view, select [Deselect Working Set] from the toolbar menu (∇) in the [C/C++ Projects] view.

### Creating a working set

- (1) Select [Select Working Set...] from the toolbar menu (∇) in the [C/C++ Projects] view to display the [Select Working Set] dialog box (shown above).
- (2) Click the [New...] button.  
This launches the [New Working Set] wizard.

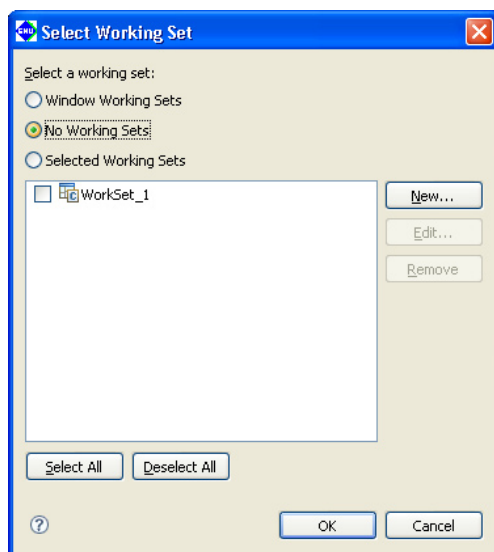


- (3) Select [C/C++] from the [Working set type:] list and click [Next>].



- (4) Enter the name of a working set in the [Working set name:] text box.
- (5) To select all resources in a project, select the check box corresponding to the project directory.  
To select for display one or more specific resources in the project directory, click the [+] icon of the project to list the resources contained in the directory and select the desired resource.
- (6) Click the [Finish] button.



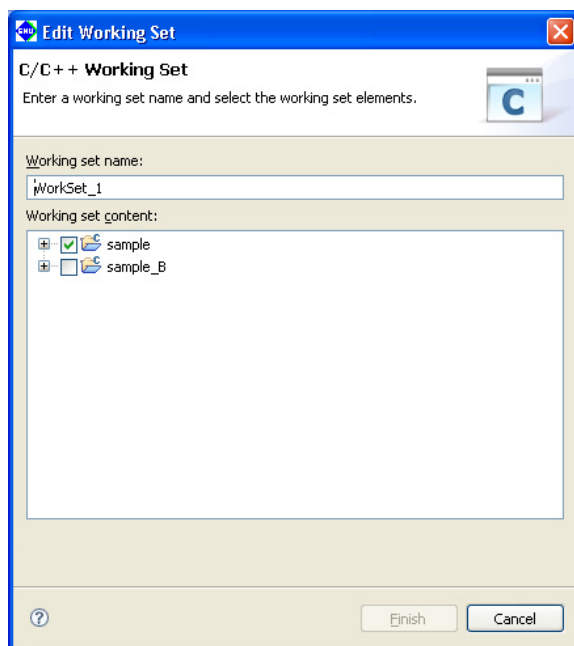


(7) Click [OK]. Your settings will be applied to the view. Clicking [Cancel] here will discard the settings.

### Editing a working set

You can modify the resource configuration of a created working set as described below.

- (1) Select [Select Working Set...] from the toolbar menu (∇) in the [C/C++ Projects] view to display the [Select Working Set] dialog box (shown above).
- (2) Select the working set you want to edit from the list and click [Edit...].  
This will display the [Edit Working Set] dialog box.



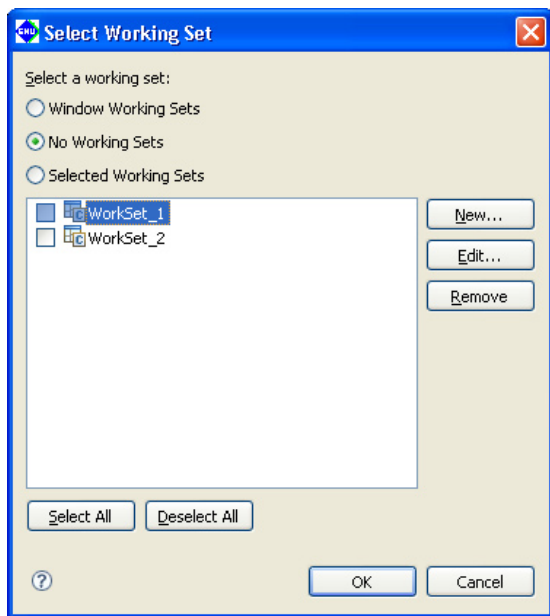
- (3) Change the selected resources for the working set just as you did when creating a new working set. Click [Finish].
- (4) Click [OK] to apply the settings to the view. Clicking [Cancel] here will discard these settings.

To edit the working set currently selected in the view, select [Edit Active Working Set...] from the toolbar menu (∇) in the [C/C++ Projects] view to display the [Edit Working Set] dialog box. When you close the [Edit Working Set] dialog box, any changes made will be directly reflected in the view.

## Deleting a working set

Delete any unnecessary working sets as described below.

- (1) Select [Select Working Set...] from the toolbar menu (∇) in the [C/C++ Projects] view to display the [Select Working Set] dialog box, showing a list of the working sets created.

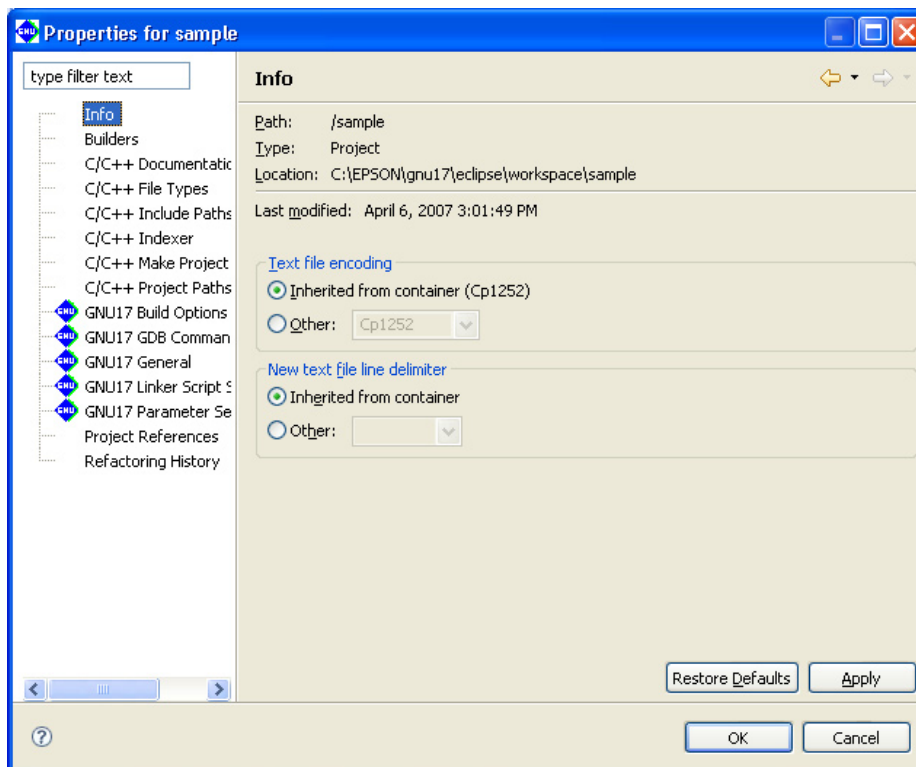


- (2) Select a working set to delete from the list and click [Remove].  
No working sets are selected in the display in this view.
- (3) Click the [OK] button.  
To use another working set, select one from the list before clicking [OK].

## 5.4.11 Project Properties

Each project has various properties that can be referenced and configured in the [Properties] dialog box. Do the following to open the [Properties] dialog box:

- (1) Select a project in the [C/C++ Projects] or [Navigator] view.
- (2) Do one of the following:
  - Select [Properties] from the [Project] menu.
  - Select [Properties] from the context menu in the [C/C++ Projects] or [Navigator] view.



Click the desired item in the properties list to the left of the dialog box to display the content set for the selected item. Make changes, if necessary.

The following properties are listed:

### 1. Info

Shows the location of the project directory. You can also set the encoding format for text files such as source files and the line delimiter.

### 2. Builders

Register or select the builder to build projects.

### 3. C/C++ Documentation

Select the help documents to be used for the project. In most cases, this setting should be left unchanged.

### 4. C/C++ File Types

Define the name or extension of the file you want handled as a C resource. In most cases, this setting should be left unchanged.

### 5. C/C++ Include Paths & Symbols

Define the path by which to search for an include file or the symbol for a preprocessor. In most cases, this setting should be left unchanged.

## 5 GNU17 IDE

### 6. C/C++ Indexer

Make settings for the indexer used in the C search or content assist.  
In most cases, this setting should be left unchanged.

### 7. C/C++ Make Project

Set make conditions.

### 8. C/C++ Project Paths

Set the destination, etc. to which the source directory and generated files are to be output.

### 9. GNU17 Build Options

Set the command line options for the compiler, assembler, and linker.

### 10. GNU17 GDB Commands

Edit a debugger startup command file.

### 11. GNU17 General

Select the target processor and memory model.

### 12. GNU17 Linker Script Settings

Edit the linker script.

### 13. GNU17 Parameter Settings

Edit the parameter file used in the debugger.

### 14. Project References

Select other projects to be referenced by the current project.

### 15. Refactoring History

Displays the refactoring history.

For more information, refer to Section 5.10.1 or the various sections that discuss the corresponding specific topic.

## 5.5 The Editor and Editing Source Files

The **IDE** incorporates editors to allow users to create and edit source files. The **IDE** can also be set to launch an external editor for source file editing.

### 5.5.1 Starting the Editor

#### Types of editors

The **IDE** provides three kinds of editors. Use the editor appropriate for the file type (file name extension).

##### 1. C editor

Used to create and edit C sources (\*.c) and header files (\*.h). This editor highlights C reserved words, comments, and strings. The editor provides a "content assist" feature that allows you to enter C reserved words or code templates by selecting from a list.

The documents opened in this editor are represented in the [C/C++ Projects] or [Navigator] view by the icons shown below.



##### 2. Assembler editor

Used to create and edit assembler sources (\*.s). This editor highlights labels, directives, and register names.

Documents opened in this editor are represented in the [C/C++ Projects] or [Navigator] view by the icons shown below.



##### 3. Text editor

Used to create and edit text files in formats (file name extensions) other than the above.

The documents opened in this editor are represented in the [C/C++ Projects] or [Navigator] view by the icons shown below.



The file types associated with the C editor or assembler editor (i.e., launch the C or assembler editor by default when opened) are registered as C project resource files. For information on registered file types, check [C/C++ File Types] in the [Properties] dialog box (displayed by selecting [Properties] from the [Project] menu) or [C/C++ > [File Types] in the [Preferences] dialog box (displayed by selecting [Preferences...] from the [Window] menu).

#### \* Word/Excel files and batch files

Eclipse, the platform on which the **IDE** is designed to run, supports OLE documents. This means that opening a "\*.doc" or "\*.xls" file will launch the specific application associated with that file type in Windows (Word or Excel), allowing the file to be edited in the **IDE**'s editor area.

Note that opening an OLE document puts it in editing mode. You will be prompted to save your changes when you close the document, even if no changes were made.

Double clicking on a "\*.cmd" or "\*.bat" file launches Command Prompt. To edit a file of one of these types, drag and drop it on the editor area.

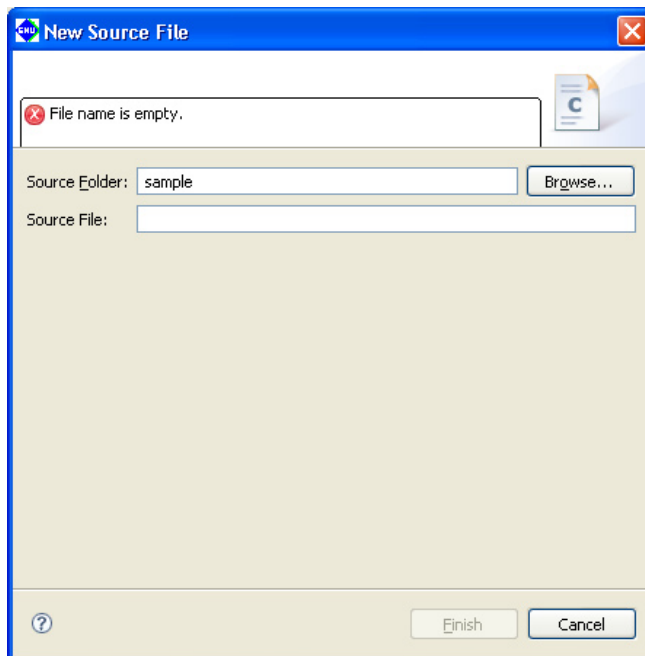
## Creating a new source

Do the following to create a new source (text) file:

(1) Do one of the following:

- Select [New] > [Source File] from the [File] menu.
- Select [New] > [Source File] from the context menu in the [C/C++ Projects] or [Navigator] view.
- Select [Source File] from the [New] shortcut in the toolbar.
- Select [Source File] from the [New C/C++ Source File] shortcut in the toolbar.
- Click the [New C/C++ Source File] button in the toolbar.

This displays the [New Source File] dialog box.



(2) Enter the name of the file you want to create in the [Source File:] text box. To create a C source or assembler source, be sure to add the file name extension appropriate for the source to be created.

(3) Click the [Finish] button.

A blank document is opened in the editor area.

## Opening a file






To open a file in the editor, double-click on the file name (or icon) in the [C/C++ Projects] or [Navigator] view.

## 5.5.2 Basic Editing Facilities

The editor is used in the same way as a general editor and offers the same general functions.

Listed below are typical editing functions and menu commands.

Table 5.5.2.1 Basic editing commands

Editing facilities	Menu bar (key shortcut)	Context menu (view that shows the menu)	Button/other operation
Create a new file	[File]>[New]>[Source File], [Header File], [File]	[New]>[Source File], [Header File], [File] (C/C++ Projects, Navigator)	 , 
Open a file	[File]>[Open File...]	[Open], [Open With] (C/C++ Projects, Navigator)	Double-click on a file name in [C/C++ Projects]/[Navigator] view
Close a file	[File]>[Close] (Ctrl+W)	–	Click on the  button in the editor tab
Close all files	[File]>[Close All] (Ctrl+Shift+W)	–	–
Save a file	[File]>[Save] (Ctrl+S)	[Save] (Editor)	
Save under another name	[File]>[Save As...]	–	–
Save all	[File]>[Save All] (Ctrl+Shift+S)	–	–
Revert to previously saved version	[File]>[Revert]	[Revert File] (Editor)	–
Print	[File]>[Print...] (Ctrl+P)	–	
Undo	[Edit]>[Undo Typing] (Ctrl+Z)	[Undo Typing] (Editor)	–
Redo	[Edit]>[Redo Typing] (Ctrl+Y)	–	–
Cut	[Edit]>[Cut] (Ctrl+X)	[Cut] (Editor)	–
Copy	[Edit]>[Copy] (Ctrl+C)	[Copy] (Editor)	–
Paste	[Edit]>[Paste] (Ctrl+V)	[Paste] (Editor)	–
Delete	[Edit]>[Delete] (Delete)	–	–
Select all	[Edit]>[Select All] (Ctrl+A)	–	–
Find	[Edit]>[Find/ Replace...] (Ctrl+F)	–	–
Replace	[Edit]>[Find/ Replace...] (Ctrl+F)	–	–
Find next	[Edit]>[Find Next] (Ctrl+K)	–	–
Find previous	[Edit]>[Find Previous] (Ctrl+Shift+K)	–	–

### 5.5.3 Editing Functions for C Source Files

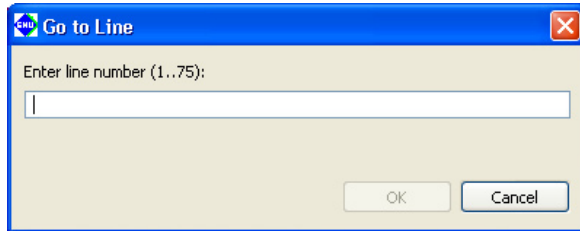
In addition to the basic editing functions shown above, the C editor provides features specific to C source code.

#### Jump to a specified line

This function allows you to specify a line number and jump to that line. The procedure is described below.

- (1) Select [Go to Line...] from the [Navigate] menu.

Displays the [Go to Line] dialog box.



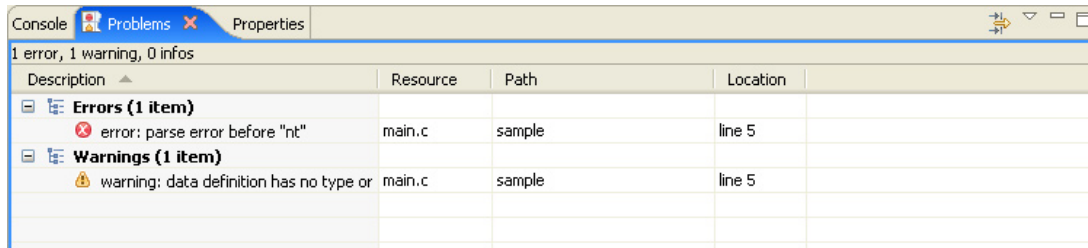
- (2) Enter a line number in the text box and click [OK].

Control will jump to the specified line.

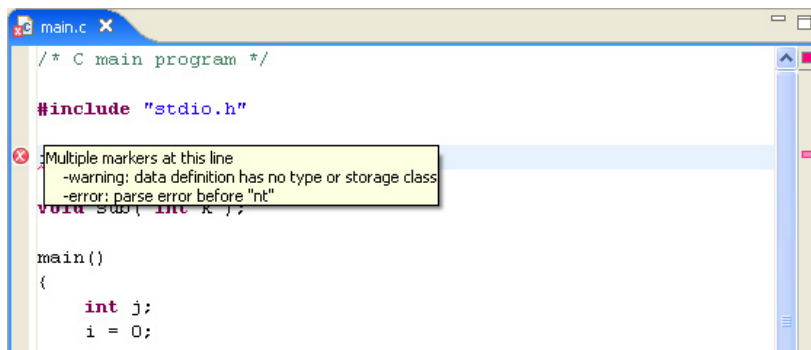
With its default settings, the C editor will not display line numbers in the editor area. Refer to Section 5.5.8 for information on enabling display of line numbers.

#### Jump to a line with an error

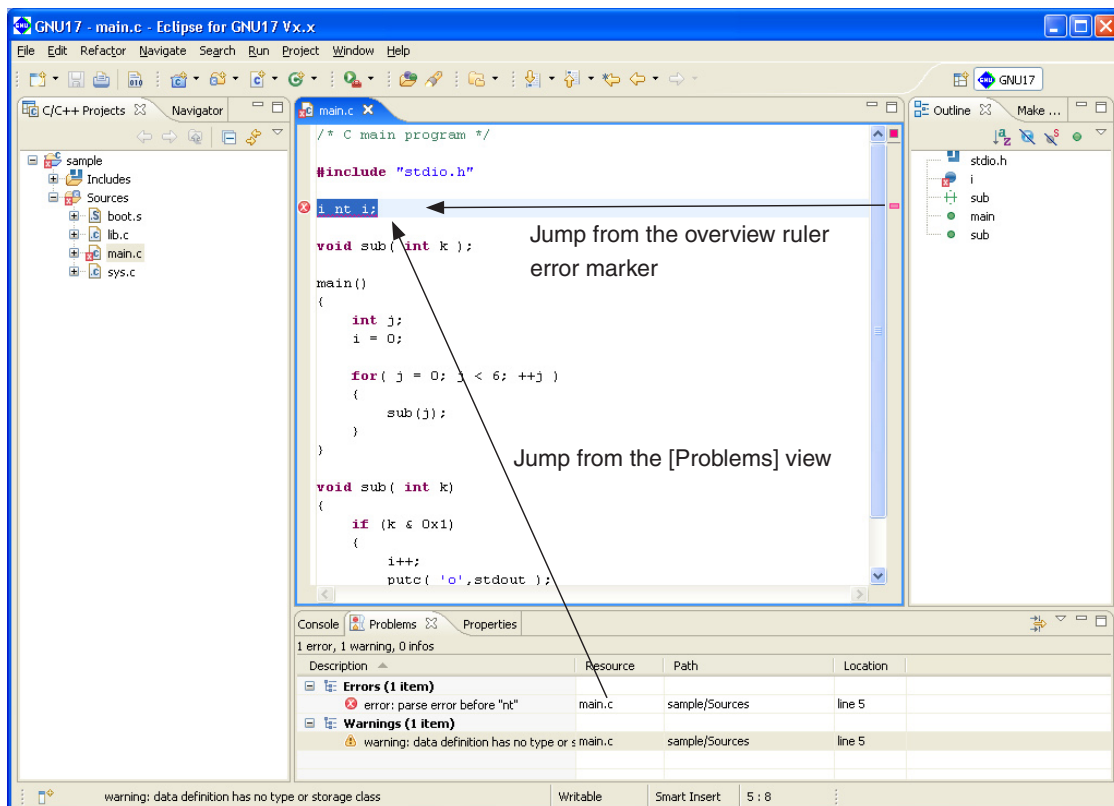
When an error occurs during a build process, a list of errors and their locations are displayed in the [Problems] view.



If an error occurs in the source file currently open in the editor, the line with the error is marked with a symbol (⊗). Hover the mouse cursor over this marker to see the nature of the error.







The [Problems] view links to the editor. Double-clicking on the compile error displayed in the [Problems] view will control jumps to the relevant line in the editor. (If closed, the file is opened.) This applies only when the error information in [Problems] view indicates the source file and the lines in error.

Although assembler sources also support jumps from the [Problems] view, C source files allow navigation through the lines with the errors from within the editor.

#### Jump back to a line with an error

If you do either of the following, control jumps back to the line containing the error closest to the current cursor position:

- Select [Next Annotation] from the [Navigate] menu.
- Click the [Next Annotation] button in the toolbar.

#### Jump forward to line containing the error

If you do either of the following, control jumps forward to the line containing the error closest to the current cursor position:

- Select [Previous Annotation] from the [Navigate] menu.
- Click the [Previous Annotation] button in the toolbar.

The editor's overview ruler (to the right of the vertical scroll bar) displays a marker to indicate the position of the error within the overall file. Click on the marker to jump to the line containing the error.

## Incremental search

You can use the incremental search function in the C editor to locate a string. Each time you enter a search string, the search results reflect the entry of each character in real time. Procedures for use are described below.

(1) Activate the file you want to search (bring it before all other files) in the editor.

(2) Select [Incremental Find Next] or [Incremental Find Previous] from the [Edit] menu.

You will see "Incremental Find" displayed in the status bar (at the bottom) of the window, with the editor placed in incremental search mode.

(3) Enter a search string.

Although no search strings are entered at the cursor position, note that the string searches proceed backward from the current position for [Incremental Find Next] or searched forward from the current position for [Incremental Find Previous].

Among the strings matching the search string, the one closest to the current position will be displayed in inverse video. The search result changes each time you enter one character. If you enter a character inadvertently, simply use the [Backspace] key to delete it. The search string you entered appears in the status bar.

Example: Entered as main

```
/* main.c          */
/* C main program */
:
[m] key entered
:
/* main.c          */
/* C main program */
:
[a] [i] [n] key entered
:
/* main.c          */
/* C main program */
:
```

**Note:** The arrow keys [←] and [→] and the [Enter] and [Esc] keys terminate incremental search mode. Be careful to avoid pressing these keys inadvertently before the end of the search.

(4) The closest occurrence of the search string is displayed after you enter the target search string.

You can use the arrow keys [↑] or [↓] to move to the next or previous occurrence. Selecting [Incremental Find Next] or [Incremental Find Previous] from the [Edit] menu has the same action.

```
/* main.c          */
/* C main program */
:
[↓] key entered
:
/* main.c          */
/* C main program */
:
```

**Note:** If no matching strings are found in the chosen search direction when you select [Incremental Find Next] or [Incremental Find Previous], the message "<string> not found" is displayed in the status bar. Even in this case, since it is possible that matching strings will exist in the document, use the arrow keys [↑] or [↓] to search the document forward or backward.

(5) To quit incremental search mode, press the [Enter] or the [Esc] key.

## Indentation

In C source files, you can shift multiple lines of code to the left or right by one tab stop. Use this function to adjust indents (for example, to align nesting in a loop statement after copying and pasting). The procedure is described below.

- (1) Select the line whose indent you want to change by placing the cursor within the line. To select multiple lines, drag across the lines in question.
- (2) Select [Shift Right] or [Shift Left] from the [Edit] menu or the editor's context menu.

Example:

```
main()
  {      ← [Shift Left] executed
int j;   ← [Shift Right] executed
  ↓
main()
  {
    int j;
```

All selected lines are moved to the left or right by one tab stop.

When shifted to the right, a tab stop is inserted at the beginning of the line(s).

If the line is shifted to the left, one tab stop is removed from the beginning of the line(s). If indented by a space, a space equal to the currently set tab size (with initial settings, four characters) is removed. For example, if a line is indented by a blank space equal to six characters when the tab stop is set to four characters, the line will have a space equal to two characters left at the beginning when shifted left. If indented by a blank of space less than four characters, shifting the line(s) to the left will have no effect.

Select [Window] > [Preferences...] to modify tab sizes in the [Preferences] dialog box ([Text Editors] settings of [General] > [Editors]).

## Commenting out specified lines

In C sources, you can comment out multiple lines, then later undo the action.

The comment-out' procedure is described below.

- (1) Select the line you want to comment out by placing the cursor at that position in the editor. To select multiple lines, drag and select the lines in question.
- (2) Select [Comment] from the editor's context menu.

Example:

```
for (j=0 ; ; j++)
  {
    disp_j();      ← [Comment] executed
    sub(j);
  }
  ↓
for (j=0 ; ; j++)
  {
//      disp_j();
      sub(j);
  }
```

All selected lines will be preceded by "//". Even if the selected line has already been turned into a comment, "//" is inserted unconditionally (whether marked with // or /\*).

Described below is the procedure for uncommenting a line previously commented out.

- (1) Select the line you want to uncomment by placing the cursor at that position in the editor. To select multiple lines, drag and select the lines in question.
- (2) Select [Uncomment] from the editor's context menu.

Example:

```

for (j=0 ; ; j++)
{
//          disp_j(); ← [Uncomment] executed
    sub(j);
}
↓
for (j=0 ; ; j++)
{
    disp_j();
    sub(j);
}

```

The "/" inserted at the beginning of the line is removed. Even when "/" is preceded by a space or tab stop, only "/" is removed. The space or tab stop left intact.

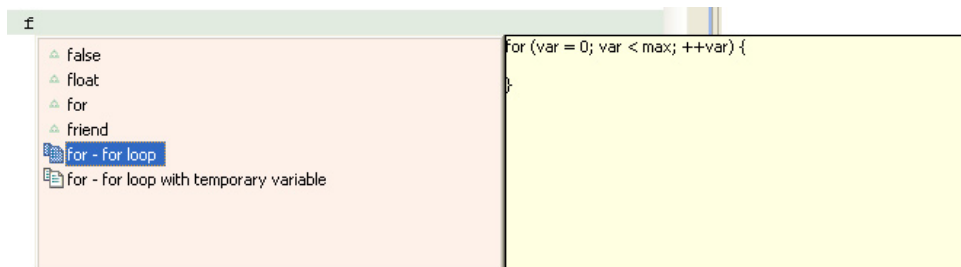
This operation does not affect comment lines beginning with "/\*".

## Content assist

The C editor has a content assist facility that allows the user to select a C reserved word or template for insertion at the text cursor position from a list as the user begins typing it. This feature is described below.

- (1) Place the text cursor at the position where you want to insert a new statement.
- (2) If you know the code you want to enter, enter the first one or two characters. This narrows the list of suggestions.  
Example: To write a for statement, enter the letter 'f'.

- (3) Select [Content Assist] from the [Edit] menu or the editor's context menu.



- (4) Choose a reserved word or template from the list and double-click.

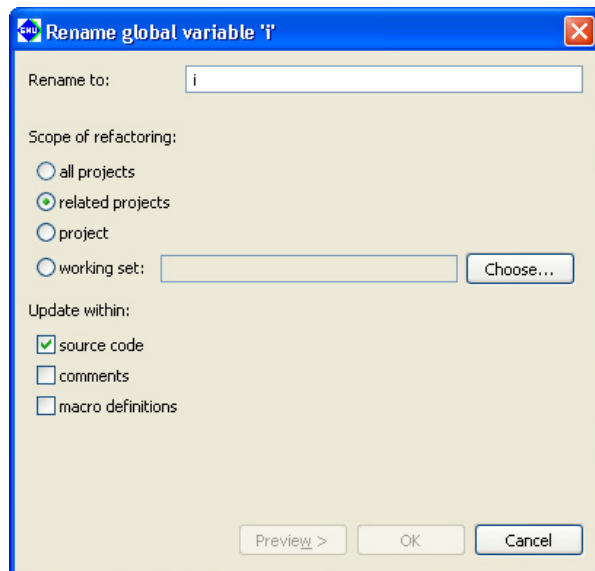
The selection is inserted at the cursor location.

Consisting of a loop statement or condition statement in fixed format, templates are listed with document icons in the left-side column. Template contents are displayed in the right-side column when you click to select it from the list. While general-purpose templates are predefined, you can also define custom templates. (Refer to [C/C++] > [Editor] > [Templates] in Section 5.9, "Customizing the IDE (Preferences)".)

## Refactoring

You can change the names of variables, types, or functions by including their declared locations and all referenced locations. The procedure is described below.

- (1) Select an element whose name you want to change from the editor or from the [C/C++ Projects] or [Outline] view.
- (2) Select [Rename...] from the [Refactor] menu. Or display the context menu of the selected element and select [Refactor] > [Rename...] from it.  
This displays the [Rename] dialog box.

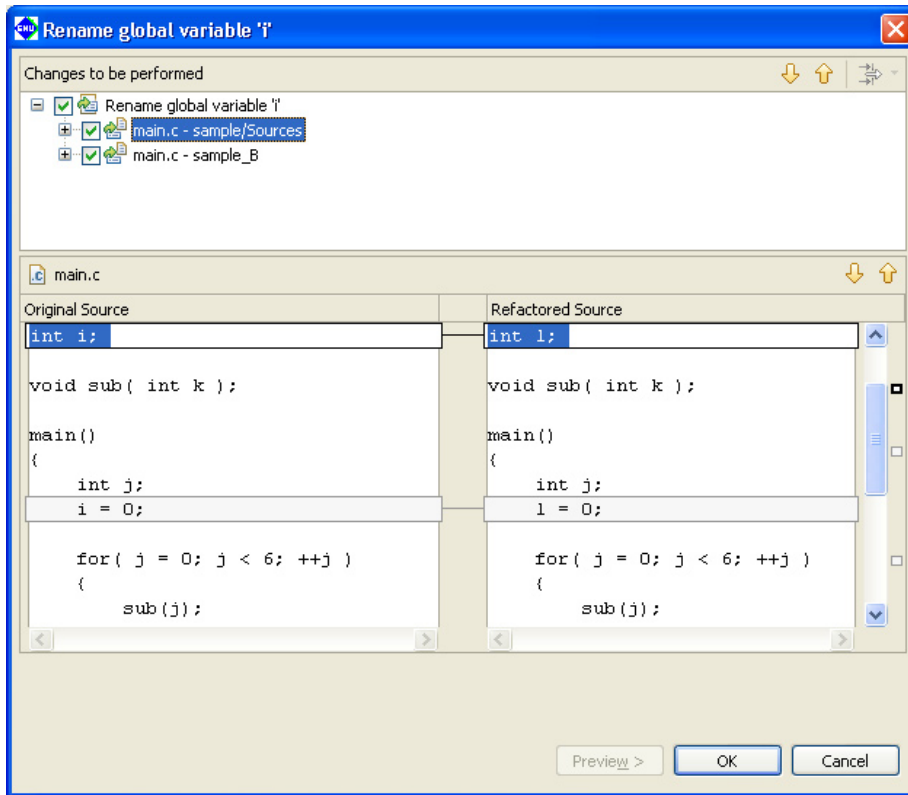


- (3) Enter a new name in the [Rename to:] text box.
- (4) Select the scope of renaming using a radio button in the [Scope of refactoring:] field.
 

[all projects]	All the opened projects will be in the scope of renaming.
[related projects]	All the projects related to the project being currently edited will be in the scope of renaming.
[project]	The project being currently edited only will be in the scope of renaming.
[working set:]	Projects or files in a working set will be in the scope of renaming. Use the [Choose...] button to select the working set.
- (5) Select the effective range using the check boxes in the [Update within:] field.
 

[source code]	The elements in source code will be renamed.
[comments]	The string in comments will be replaced.
[macro definitions]	The elements in macro definitions will be renamed.

- (6) Click the [Preview>] button to display a list of the corresponding resources. If you do not want to change any resource, deselect it by removing a check mark.



Use the arrow buttons to check the locations of the elements that will be renamed.

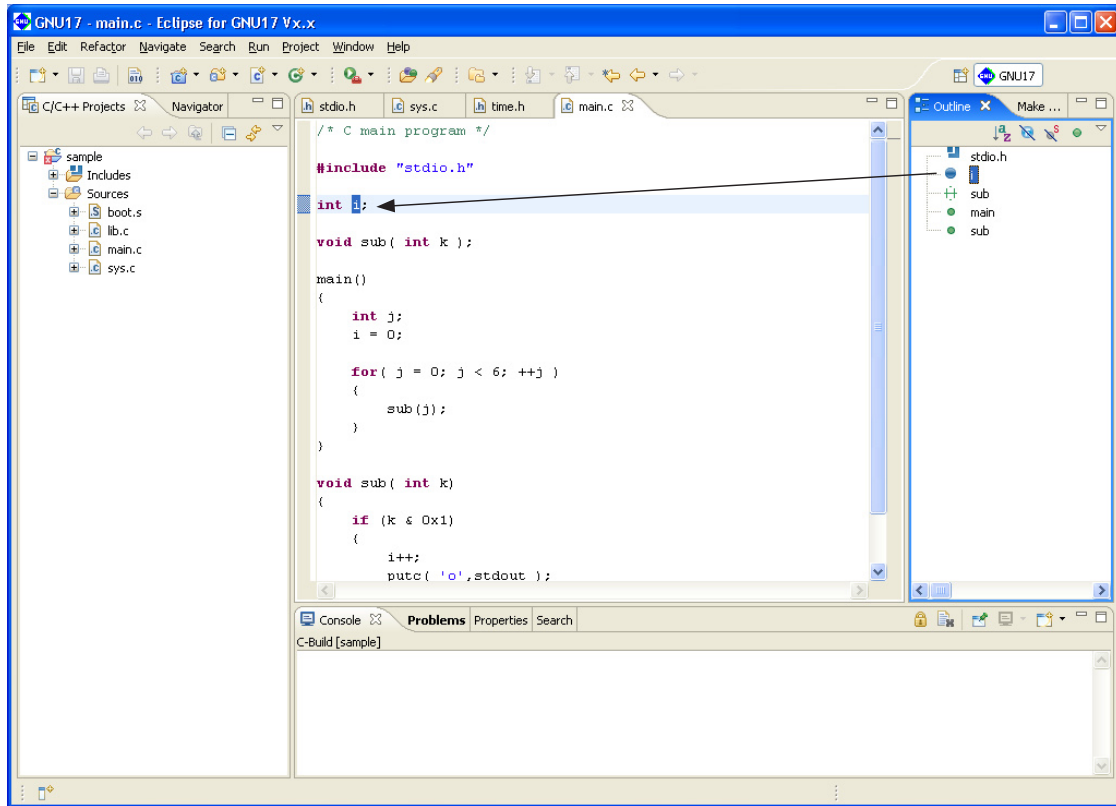
- (7) Click the [OK] button to change, or click the [Cancel] button to cancel.

To undo the change after completing this operation, select [Undo .....] from the [Refactor] menu, or select [Refactor] > [Undo .....] from the context menu of the selected element. After performing an Undo, [Redo .....] in the menu will be effective, allowing you to repeat the operation canceled by Undo.

## 5.5.4 [Outline] View

The [Outline] view shows the variables and functions defined in the C source currently in front of all other files in the editor area.

Click on a variable or function name to jump to its defined position in the source.



## 5.5.5 Navigation History

The IDE editor retains a history of the files opened previously, making it possible to trace the history backward or forward, as with a Web browser. You only navigate through a history, and cannot change the edited content.

### Tracing a history backward

Do one of the following:

- Select [Back] from the [Navigate] menu.
- Click the [Back] button in the toolbar.

These operations will return you to the immediately preceding point in a history.

Click [▼] to the right of the [Back] button in the toolbar to display a list of files in a history. If you wish, you can select a file from this list.

### Tracing a history forward

Do one of the following:

- Select [Forward] from the [Navigate] menu.
- Click the [Forward] button in the toolbar.

These operations will move you forward to the point immediately following in a history.

Click [▼] to the right of the [Forward] button in the toolbar to display a list of files in a history. If you wish, you can select a file from this list.

### Jumping to a location just edited

This feature allows you to return to the source line you last edited. Do one of the following:

- Select [Last Edit Location] from the [Navigate] menu.
- Click the [Last Edit Location] button in the toolbar.

These operations will always move you to the point just edited until you choose to edit another location. Any entry made in the document is judged as editing. Deleting the characters entered or undoing an operation does not reverse the history. If the last edit made was a line deletion, the history will go to the point preceding the deletion.



## 5.5.6 Bookmarks

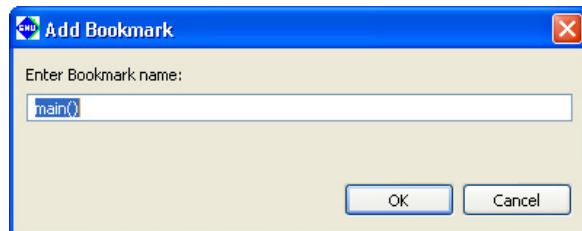
Frequently examined points (lines) can be marked by bookmarks. Lines marked with a bookmark are listed in [Bookmarks] view to allow users to move rapidly to those locations.

### Attaching a bookmark

Do the following to attach a bookmark:

- (1) Place the cursor at the source line where you want to attach a bookmark.
- (2) Do one of the following:
  - Select [Add Bookmark...] from the [Edit] menu.
  - Right-click on the editor's marker bar (the left edge of the editor area) to display the context menu, then select [Add Bookmark...].

This displays the [Add Bookmark] dialog box.



- (3) Set a bookmark name. Use the name displayed in the [Enter Bookmark name:] text box unchanged, or enter another name and click [OK].

A bookmark marker appears in the editor's marker bar.



Open the [Bookmarks] view. The bookmark just set has been added to the list.

Description	Resource	Path	Location
Sub Function	main.c	sample	line 9
Sub Function	main.c	sample	line 20

You can rename a bookmark simply by clicking in the [Description] column.

## Jumping to a bookmark

You can jump from the [Bookmarks] view to a source line marked by a bookmark.

- (1) Activate the [Bookmarks] view. If not open, select [Show View] > [Bookmarks] from the [Window] menu.
- (2) Do one of the following:
  - Double click in the line of the desired bookmark.
  - Right-click anywhere in the line of the desired bookmark to display the context menu, then select [Go To].

The editor will jump to the bookmark position.

## Removing a bookmark

If a bookmark becomes unnecessary, you can remove it in the editor or from the [Bookmarks] view.

### Performing deletions in the editor

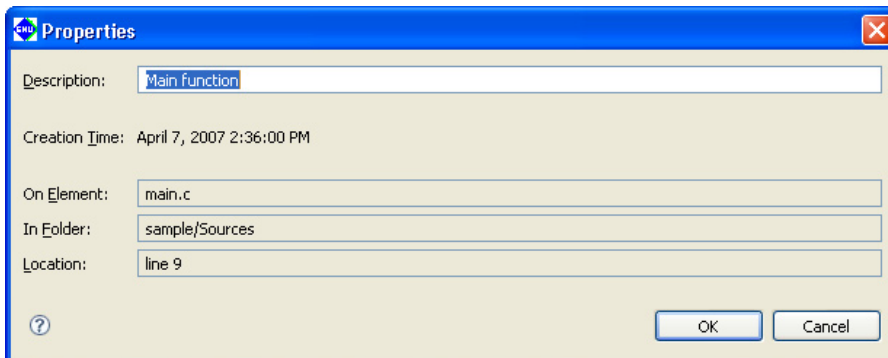
Right-click on a bookmark marker you want to remove to display the context menu, then select [Remove Bookmark].

### Removing in the [Bookmarks] view

- (1) Click a bookmark marker to select it for removal.
- (2) Do one of the following:
  - Click the [Delete] button in the toolbar of the view.
  - Display the context menu and select [Delete] from it.

## Showing bookmark information

Right-click anywhere in the line of the desired bookmark to display the context menu, then select [Properties]. This displays the [Properties] dialog box, showing the date and time of creation, in addition to the information shown in the view.



You also rename the bookmark.

## Filtering and sorting the bookmark list

If the number of bookmarks makes the list in the [Bookmarks] view unwieldy, you can choose to hide certain bookmarks or sort the bookmarks by item.

### Filters

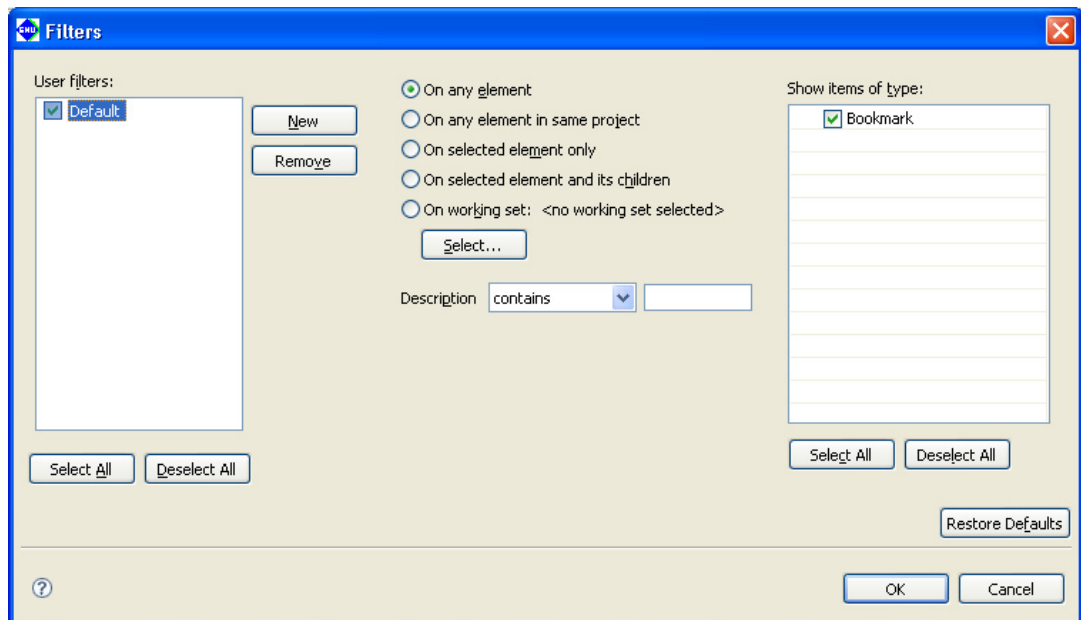
Use filters to display only the desired bookmarks and to hide other bookmarks.

Furthermore, two or more filters can be configured and used as necessary.

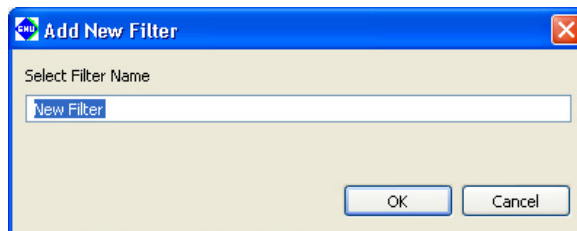
A new filter can be configured as in the procedure below.

- (1) Activate [Bookmarks] view.
- (2) Do one of the following:
  - Click the [Configure Filters] button in the toolbar of the view.
  - Select [Configure Filters...] from the view menu (▽).

This displays the [Filters] dialog box.



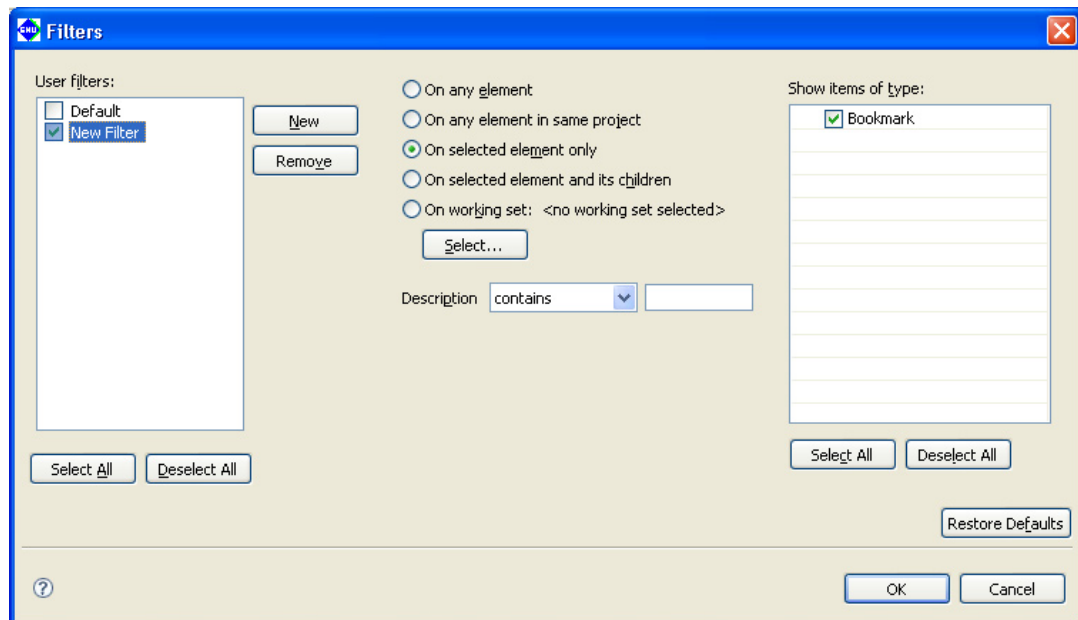
- (3) Click the [New] button.



Enter the name of the filter to be configured and click [OK].

- (4) Deselect the check boxes for the other filters displayed in [User filters:]. Select the new filter created and turn its check box on.

- (5) Select conditions to display bookmarks.
- |  |  |
|--|--|
| [On any element]                       | The bookmarks attached in all the opened projects will be displayed.   |
| [On any element in same project]       | The bookmarks attached in the project being currently selected will be displayed.  |
| [On selected element only]             | The bookmarks attached in the file that has been selected in the [C/C++ Projects]/[Navigator] view or activated in the editor will be displayed.   |
| [On selected element and its children] | The bookmarks attached in the files located in the project or folder that has been selected in the [C/C++ Projects]/[Navigator] view or in the selected file will be displayed.  |
| [On working set:]                      | The bookmarks attached in a working set will be displayed. Use the [Select...] button to select the working set.   |
| [Description] - [contains]             | In addition to the condition above, this option limits the bookmarks to be displayed to those whose [Description] contain the string entered in the text box. Leave the text box empty when this condition is not used.          |
| [Description] - [doesn't contain]      | In addition to the condition above, this option limits the bookmarks to be displayed to those whose [Description] does not contain the string entered in the text box. Leave the text box empty when this condition is not used. |



- (6) Click [OK].

The [Bookmarks] view now displays just the bookmarks meeting the specified conditions.

To change the conditions for the currently selected filter, omit Steps (3) and (4) and just select the conditions.

When two or more filters are created, display the dialog box above and select the filters to be used from the [User filters:] list. Or select them from the submenu of the [Filters] view menu.

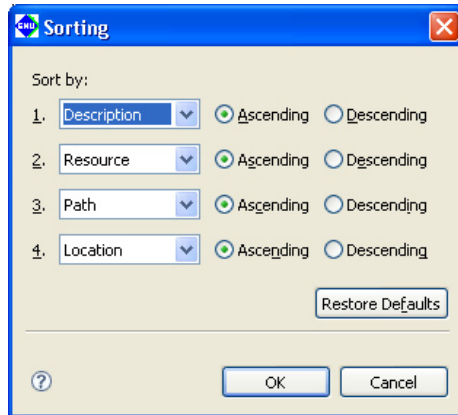
Refer to Section 5.10.5 for information on settings made in the [Filters] dialog box.

## Sorting

You can prioritize items and sort the displayed bookmarks in order of prioritized items.

- (1) Activate the [Bookmarks] view.
- (2) Select [Sorting...] from the view menu (▽).

This displays the [Sorting] dialog box.



Refer to Section 5.10.6 for information on settings made in the [Sorting] dialog box.

- (3) Select a condition and click [OK].

The [Bookmarks] view will sort the list of displayed bookmarks based on the specified condition.

## 5.5.7 Tasks

While creating a source, if you want to jump ahead while leaving part of the source pending, you can retain information on that position or the content of work recorded as tasks. Although a task in the editor is a marker for the jump destination similar to a bookmark, you can specify task the priorities and use a list of tasks displayed in the [Tasks] view as a To Do list.

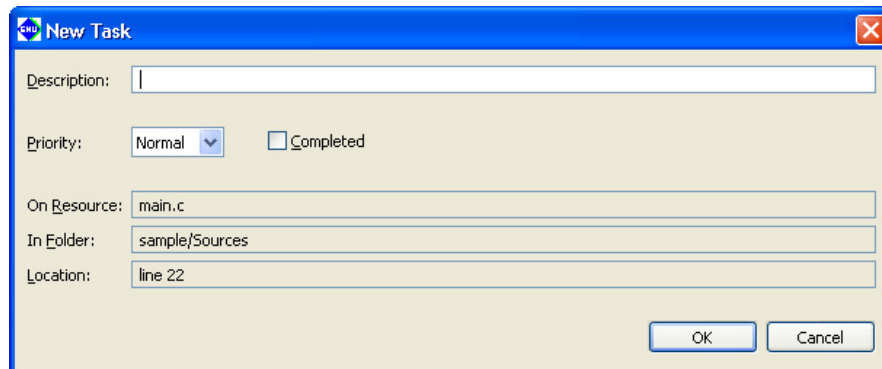
### Creating a task

Do the following to create a task:

#### When including source line information

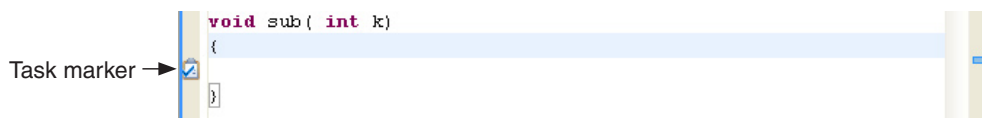
- (1) Place the cursor at the source line in which you want to set a task.
- (2) Do one of the following:
  - Select [Add Task...] from the [Edit] menu.
  - Right-click on the editor's marker bar (the left edge of the editor area) to display the context menu, then select [Add Task...].

This displays the [New Task] dialog box.

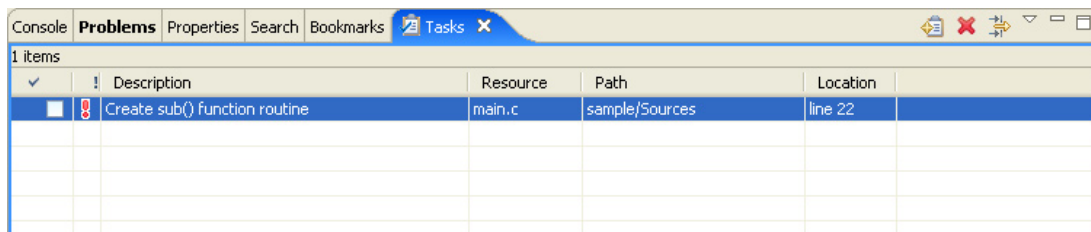


- (3) Enter a task description in the [Description:] text box.
- (4) Select priority (High, Normal, or Low) from the [Priority:] combo box.
- (5) If you want the task to be created as a completed task, select the [Completed] check box.
- (6) Click the [OK] button.

A task marker is displayed in the editor's marker bar.



Open the [Tasks] view. The task created should appear in the list.



The check box on the left edge of the list indicates whether a task is "Completed" or "Not Completed". For a completed task, click and check this box.

The column next to it indicates the task priority with an icon.

! = High, blank = Normal, ↓ = Low

Click in this column to display a pull-down list box. Use the pull-down list box to revise the task priority.

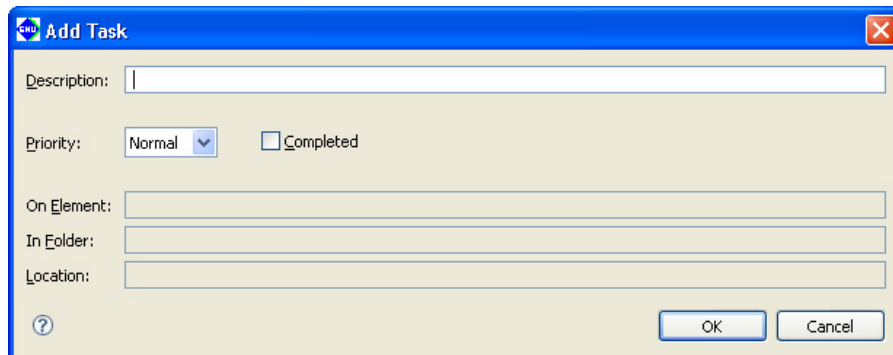
Click in the [Description] column to change a task description.

### When not including source line information

You can create a To Do item not associated with a specific source file.

- (1) Do one of the following:
  - Click the [Add Task] button in the toolbar of the [Tasks] view.
  - Select [Add Task...] from the context menu of the [Tasks] view.

This displays the [Add Task] dialog box.



- (2) Enter a description of a task in the [Description:] text box.
- (3) Select priority (High, Normal, or Low) from the [Priority] combo box.
- (4) If you want to create the task as a completed task, select the [Completed] check box.
- (5) Click the [OK] button.

In this case, no resources or line numbers are set.

### Jumping to a set task position

You can jump from the [Tasks] view to the source line in which you set a task.

- (1) Activate the [Tasks] view. If not open, select [Show View] > [Tasks] from the [Window] menu.
- (2) Do one of the following:
  - Double click in the line of the desired task.
  - Right-click in the line of the desired task to display the context menu, then select [Go To].

The editor will jump to the position at which the task is set.

### Removing a task

If a task does no longer need to be displayed, you can remove it in the editor or from the [Tasks] view.

#### Deleting in the editor

Right-click on the task marker you want to remove to display the context menu, then select [Remove Task].

#### Removing from the [Tasks] view

- (1) Click to select the task marker you want to remove.
- (2) Do one of the following:
  - Click the [Delete] button in the toolbar of the view.
  - Display the context menu and select [Delete] from it.

#### Removing completed tasks

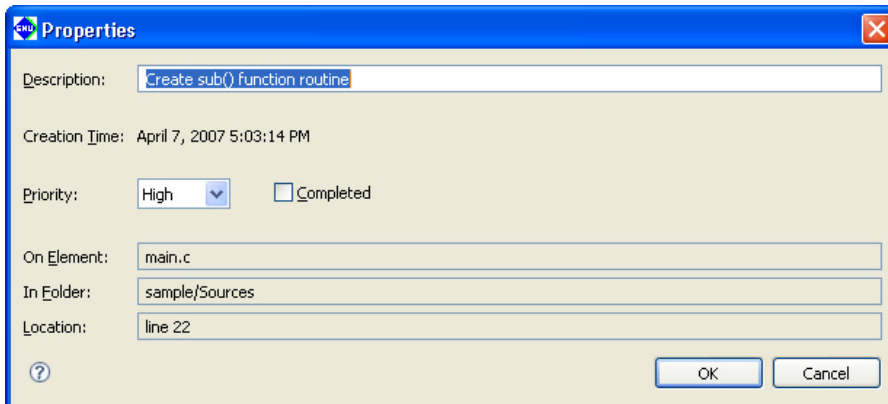
You can remove completed tasks only one at a time.

- (1) Select [Delete Completed Tasks] from the context menu of the [Tasks] view.
- (2) This displays a confirmation dialog box. Click [OK] to remove or [Cancel] to cancel.

You can also use filters to hide completed tasks without deleting them (described later).

## Showing task information

Right-click in the line of the desired task to display the context menu, then select [Properties]. This displays the [Properties] dialog box, showing the date and time of creation, in addition to the information shown in the view.



You also can alter the task description here.

## Filtering and sorting the task list

If the number of tasks makes the list in [Tasks] view unwieldy, you can choose to hide certain tasks or sort tasks by item.

### Filters

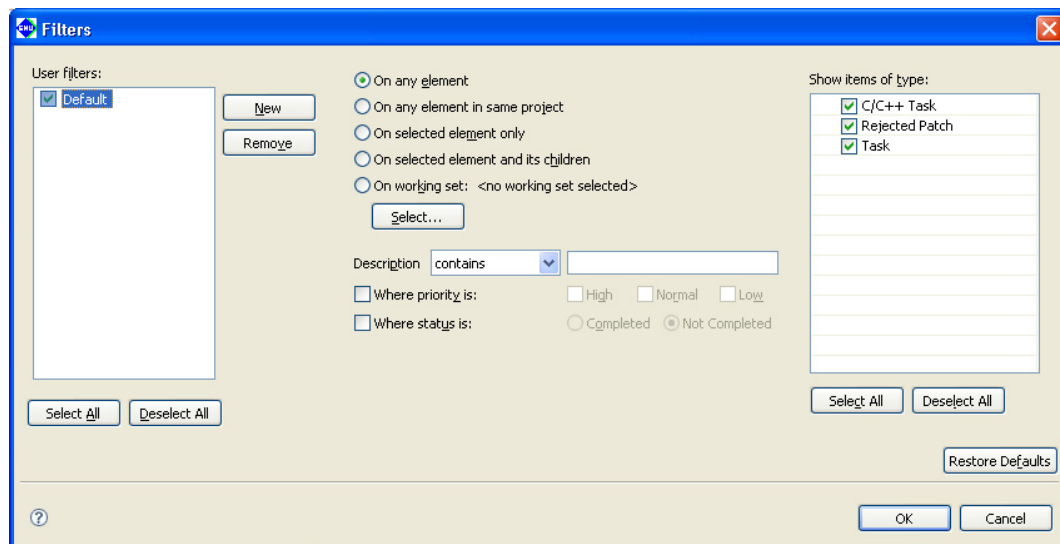
Use filters to display only the necessary tasks and to hide others.

Furthermore, two or more filters can be configured and used as necessary.

A new filter can be configured as in the procedure below.

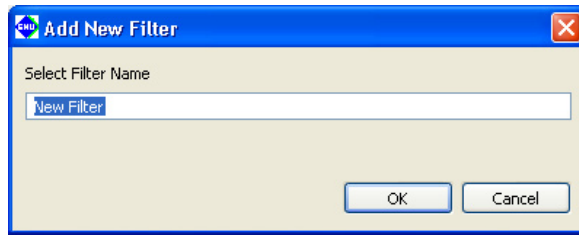
- (1) Activate [Tasks] view.
- (2) Do one of the following:
  - Click the [Configure Filters] button in the toolbar of the view.
  - Select [Configure Filters...] from the view menu (▼).

This displays the [Filters] dialog box.





- (3) Click the [New] button.



Enter the name of the filter to be configured and click [OK].

- (4) Deselect the check boxes for the other filters displayed in [User filters:]. Select the new filter created and turn its check box on.

- (5) Select conditions to display tasks.

[On any element]

The tasks set in all the opened projects will be displayed.

[On any element in same project]

The tasks attached in the project being currently selected will be displayed.

[On selected element only]

The tasks set in the file that has been selected in the [C/C++ Projects]/[Navigator] view or activated in the editor will be displayed.

[On selected element and its children]

The tasks set in the files located in the project or folder that has been selected in the [C/C++ Projects]/[Navigator] view or in the selected file will be displayed.

[On working set:]

The tasks set in a working set will be displayed. Use the [Select...] button to select the working set.

[Description] - [contains]

In addition to the condition above, this option limits the tasks to be displayed to those whose [Description] contain the string entered in the text box. Leave the text box empty when this condition is not used.

[Description] - [doesn't contain]

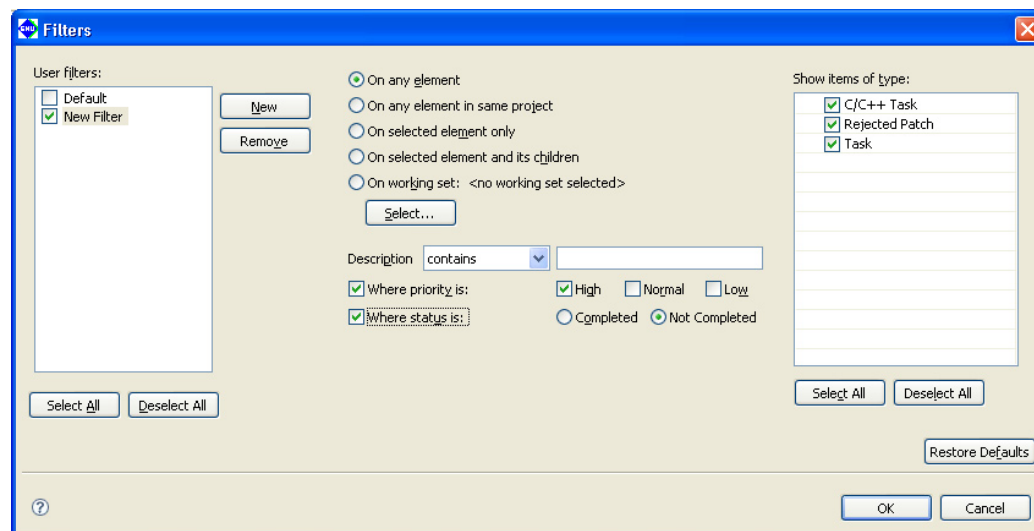
In addition to the condition above, this option limits the tasks to be displayed to those whose [Description] does not contain the string entered in the text box. Leave the text box empty when this condition is not used.

[Where priority is:] - [High/Normal/Low]

Select the check box to display tasks with the specified priority (High, Normal, or Low) only.

[Where status is:] - [Completed/Not Completed]

Select the check box and a radio button to display either the completed tasks or not completed tasks only.



(6) Click [OK].

The [Tasks] view now displays only the tasks meeting the specified condition.

To change the conditions for the currently selected filter, omit Steps (3) and (4) and just select the conditions.

When two or more filters are created, display the dialog box above and select the filters to be used from the [User filters:] list. Or select them from the submenu of the [Filters] view menu.

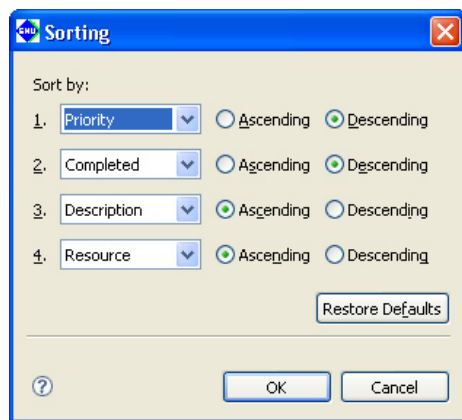
Refer to Section 5.10.5 for information on settings made in the [Filters] dialog box.

## Sorting

You can prioritize items and sort the displayed tasks in order of prioritized items.

- (1) Activate the [Tasks] view.
- (2) Select [Sorting...] from the view menu (∇).

This displays the [Sorting] dialog box.



Refer to Section 5.10.6 for a discussion of the contents set in the [Sorting] dialog box.

- (3) Select a condition and click [OK].

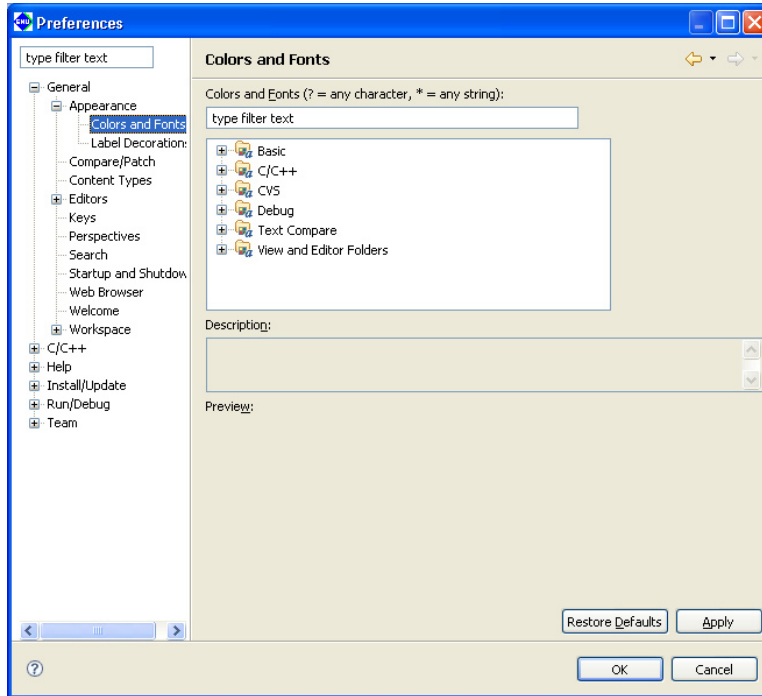
The [Tasks] view sorts the list of displayed tasks based on the condition specified.

## 5.5.8 Customizing the Editor

You can change the font and tab size used in the editor in the [Preferences] dialog box. The [Preferences] dialog box is displayed when you select [Preferences...] from the [Window] menu.

Listed below are the main pages and customization items in the [Preferences] dialog box associated with the editor. Refer to Section 5.9, "Customizing the IDE (Preferences)" for more information on the [Preferences] dialog box.

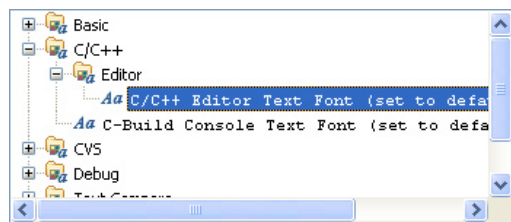
### Text fonts and colors ([General] > [Appearance] > [Colors and Fonts])



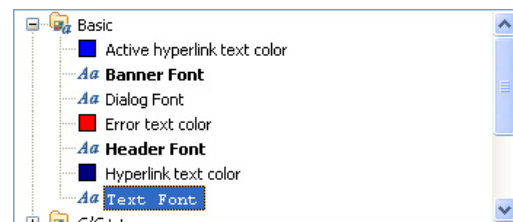
Here, you can change the default text fonts used by the C editor and assembler editor.

- (1) To change fonts for the C editor, select [C/C++] > [Editor] > [C/C++ Editor Text Font] from tree view. Select [Basic] > [Text Font] from tree view to change assembler editor text fonts.

C editor fonts



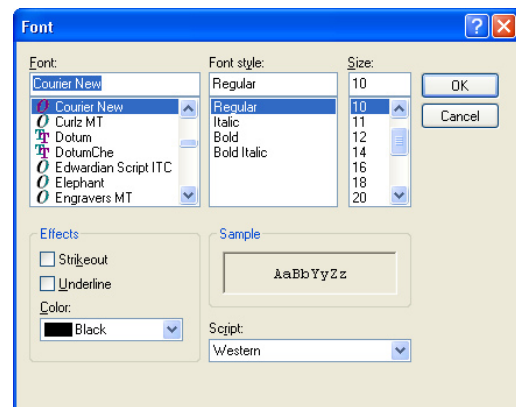
Assembler editor fonts



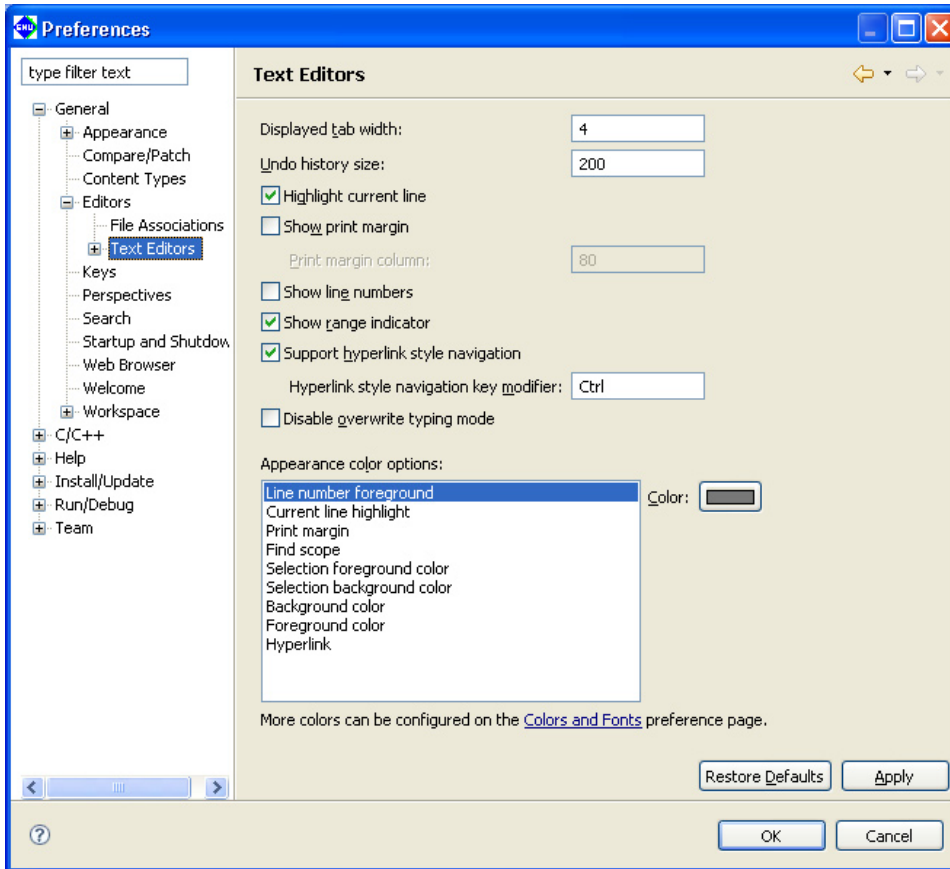
- (2) Click [Change...] to display the font select dialog box. Select a font, font style, and display color in the dialog box.

Or use the [Use System Font] button to select the standard Windows font.

- (3) Click [Apply] or [OK] to complete the settings.



## Changing the editor tab size and displaying line numbers ([General]>[Editors]>[Text Editors])



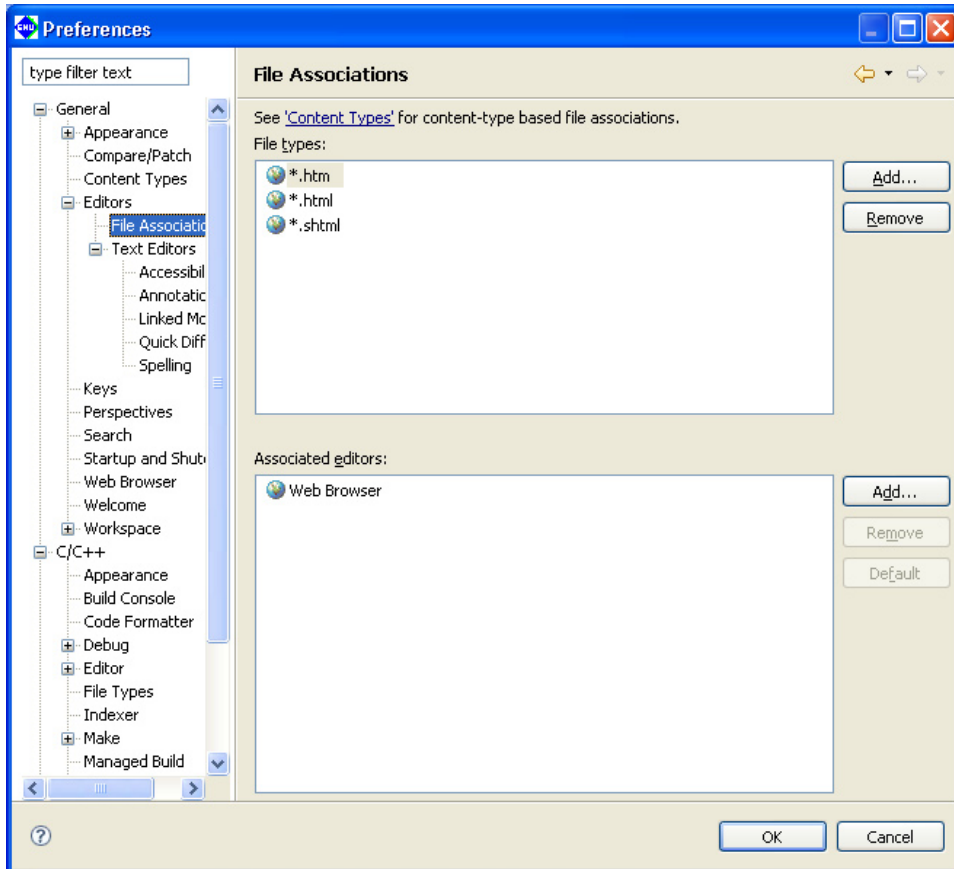
To change the tab size, set the number of characters for the tab width in the [Displayed tab width:] text box. Select the [Show line numbers] check box to enable display of line numbers.

You can also set highlighting and other options on this page.

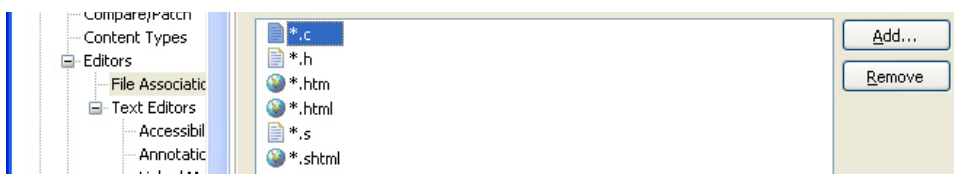
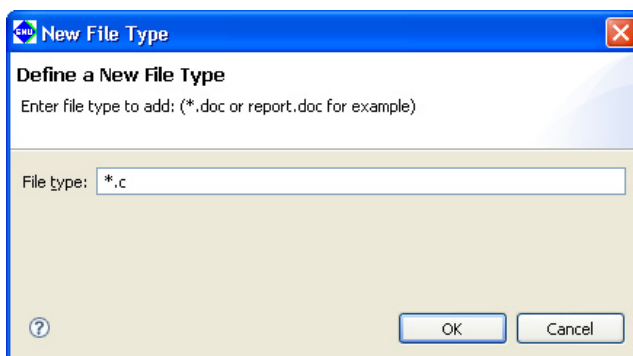
## 5.5.9 Using an External Editor

You can register a preferred editor to launch from the **IDE** for resource editing. The procedure for registering an external editor is described below.

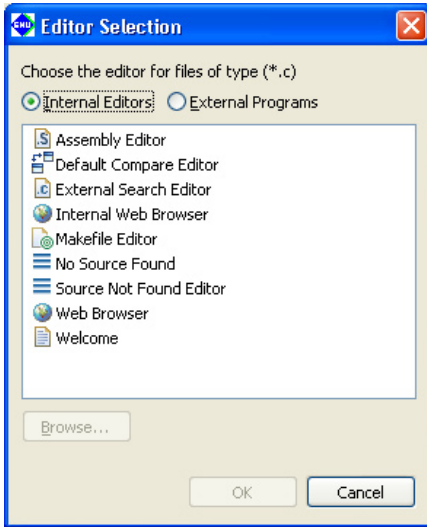
- (1) Select [Preferences...] from the [Window] menu.  
This displays the [Preferences] dialog box.
- (2) Select [General] > [Editors] > [File Associations] from the setup items listed in tree view on the left side of the dialog box.



- (3) From [File types:], select the file type (file name extension) you want to edit with the editor being registered. If the file type does not appear in the list, display the dialog box below by clicking the [Add...] button for [File types:] and enter the file name extension (\*.c, \*.h, \*.s) to add it to the list.



- (4) Click the [Add...] button for [Associated editors:].  
This displays the [Editor Selection] dialog box.



- (5) Select the [External Programs] radio button.
- (6) Select the editor you want to register from the list. If the editor does not appear in the list, click the [Browse...] button and use the select dialog box.
- (7) Click the [OK] button to close the [Editor Selection] dialog box.
- (8) Click the [OK] button to close the [Preferences] dialog box.

The file name extension you selected and the external editor have been correlated to each other by the above operation. Do this setting for all file types you want to edit.

The following describes how to open a file with the registered editor.

- (1) Select a file in the [C/C++ Projects] or [Navigator] view.
- (2) Right-click on the file to display the context menu. Select the registered editor from [Open With].

This opens the selected file in the external editor.

**Note:** You must first close files already open in the **IDE** editor before reopening them in an external editor.

## 5.6 Search

In addition to a function that can be used to search a document opened in the editor, the **IDE** incorporates search features that allow you to search text in the entire workspace or project and to set search conditions for resources or C elements. Search results are displayed in the [Search] view. This section describes how to use these search features.

### 5.6.1 Text Search

You can search for the string being selected in the editor not only in the file but also outside the file. The operation procedure is as follows:

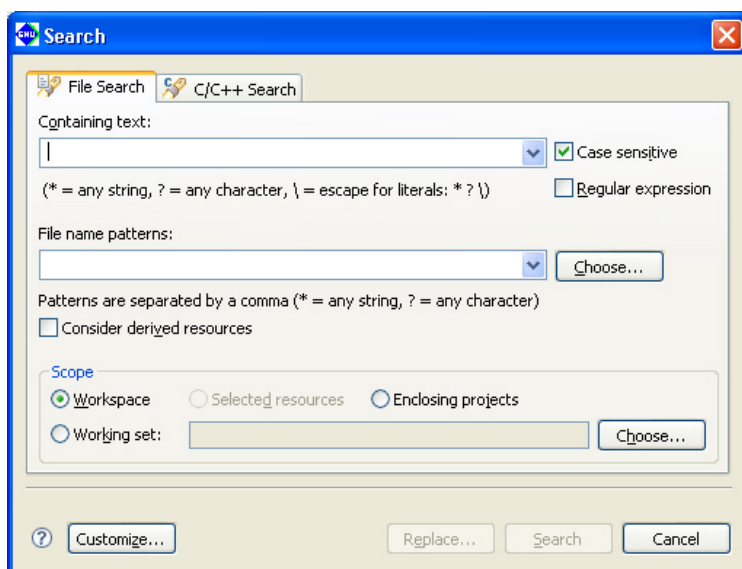
- (1) In the editor, drag and select the string to be searched.
- (2) Select [Text] from the [Search] menu and select a search domain (workspace, current project, current file or a specified working set) from its submenu.

Search results are displayed in the [Search] view.

### 5.6.2 File Search

You can search for a resource in the workspace, current project, or the specified working set. You can also search for text data included in the file being currently edited. To perform a File Search, do one of the following to display the [File Search] page of the [Search] dialog box:


- Select [File...] from the [Search] menu.
- Select [Search...] from the [Search] menu, then the [File Search] tab in the ensuing [Search] dialog box.
- Click the [Search] button in the window toolbar, then select the [File Search] tab in the ensuing [Search] dialog box.



Set the search parameters as described below and click the [Search] button. When the search ends, the search results are displayed in the [Search] view.

#### [Containing text:]

Enter the text string being searched for. To search for files only, leave this combo box blank.

If the same search was previously performed from this page, you can select it from the pull-down list. (Click  to display the pull-down list). The following are valid wildcards in the search string:

- \*: Any string
- ?: Any character
- \: Place in front of \*, ?, or \ to specify them as the search character (\\*, \?, or \\).

#### [Case sensitive]

Select this check box to make searches case-sensitive.

**[Regular expression]**

If this check box is selected, the search will be conducted matching regular expression patterns. This search mode allows you to use a regular expression input assist facility. This is described below.

- (1) Select the [Regular expression] check box.
- (2) Place the cursor in the [Containing text:] text field. A "🔍" will be displayed in front of the text field, indicating that the input assist facility is enabled.
- (3) Press the [Ctrl] + [Space] keys.
- (4) Select the syntax you want to enter from the pull-down list.

**[File name patterns:]**

Enter the file type or name pattern to search for. Separate multiple patterns with commas (.). Searches for multiple patterns assume an OR condition. Select the file types from the dialog box displayed by clicking the [Choose...] button. If the pattern you're looking for was previously entered in this page, you can select from the pull-down list. (Click ▾ to display the pull-down list).

The following are valid wildcards in the search string:

- \*: Any string
- ?: Any character

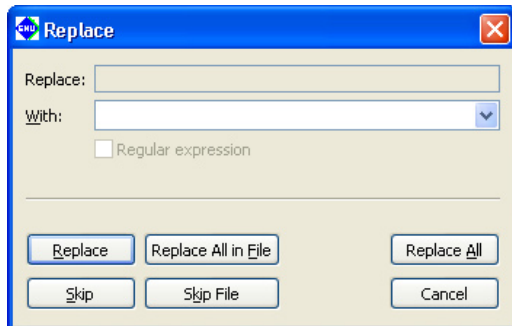
**[Scope]**

Use the radio buttons listed below to narrow the search domain:

- [Workspace]** Entire workspace
- [Selected Resources]** Resource selected in the [C/C++ Projects] or [Navigator] view
- [Enclosing Projects]** Project including the resource selected in the [C/C++ Projects] or [Navigator] view
- [Working Set:]** Resource in a selected working set. Use the dialog box displayed by clicking the [Choose...] button to select a working set.

**[Customize...]**

Selects the search page ([File Search] or [C/C++ Search]) to display in the [Search] dialog box.

**[Replace...]**

Performs a search using the parameters specified above, stopping at the first match. The matching search string is automatically selected. Use the [Replace] dialog box displayed to replace this string.

**[With:]**

Enter the replacement string.

**[Replace]**

Replaces the currently selected occurrence of the search string with the new string and begins searching for the next occurrence.

**[Replace All in File]**

Replaces all other occurrences of the search string in the current file and begins searching in the next file.

**[Replace All]**

Replaces all occurrences of the search text within the search domain specified by [Scope].

**[Skip]**

Searches for the next occurrence of the search string without replacing the currently selected occurrence of the search string.

**[Skip File]**

Searches in the next file without replacing all remaining occurrences of search text in the current file.

**[Cancel]**

Cancels replacement.

**[Search]**

Performs a search using the parameters specified above.

**[Cancel]**

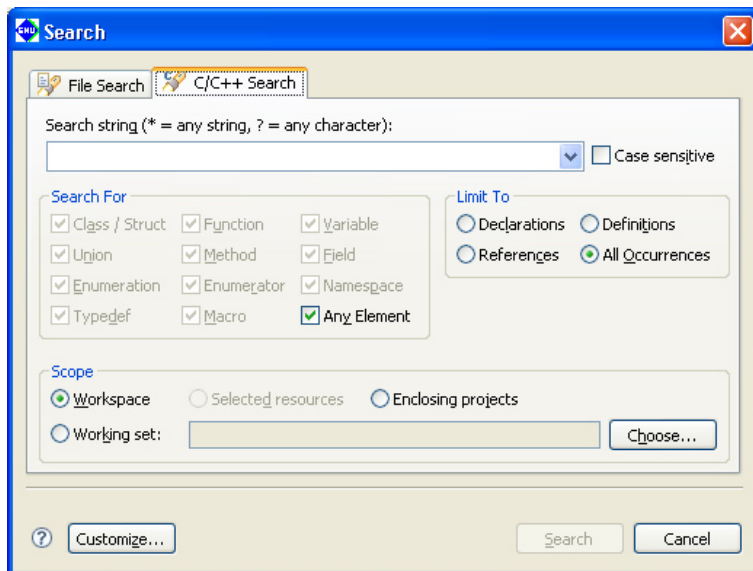
Cancels a search.



### 5.6.3 C Search

Use C Search to search for strings, function names, and other elements within workspace resources. To perform a C Search, do one of the following to display the [C/C++ Search] page of the [Search] dialog box:

- Select [C/C++...] from the [Search] menu.
- Select [Search...] from the [Search] menu.
- Click the [Search] button in the window toolbar.



Set the search parameters as described below and click the [Search] button. When the search ends, the search results are displayed in the [Search] view.

#### [Search string:]

Enter a search string or select a string in the editor before opening the [Search] dialog box.

If a previous search was made for same string, you can reselect it from the pull-down list. (Click  to display the pull-down list).

The following are valid wildcards in the search string:

- \*: Any string
- ?: Any character

#### [Case sensitive]

Select this check box to make searches case-sensitive.

#### [Search For]

Selecting one of the check boxes in this section to specify the target element to look for:

[Class/Struct]	Structure
[Function]	Global function (not including structure, and union member functions)
[Variable]	Variable (not including structure, and union members)
[Union]	Union
[Method]	Method (structure, or union members)
[Field]	Field (structure, or union members)
[Enumeration]	Enumeration
[Enumerator]	Enumerator
[Namespace]	Name space (ineffective)
[Typedef]	Type definition
[Macro]	Macro definition
[Any Element]	All elements are searched for. Selecting this check box disables the check boxes for all other elements.

**[Limit To]**

Restrict the search target by making the selections shown below.

<b>[Declarations]</b>	Declared location
<b>[Definitions]</b>	Defined place (function, method, variable, field)
<b>[References]</b>	Referenced location
<b>[All Occurrences]</b>	All occurrences, including the above

**[Scope]**

Use the radio buttons listed below to narrow the search domain:

<b>[Workspace]</b>	Entire workspace
<b>[Selected Resources]</b>	Resource selected in the [C/C++ Projects] or [Navigator] view
<b>[Enclosing projects]</b>	Project that contains the resource selected in the [C/C++ Projects] or [Navigator] view
<b>[Working Set:]</b>	Resources in a selected working set. Use the dialog box displayed by clicking the [Choose...] button to select a working set.

**[Customize...]**

Selects the search page ([File Search] or [C/C++ Search]) to display in the [Search] dialog box.

**[Search]**

Performs a search using the parameters specified above.

**[Cancel]**

Cancels a search.

### 5.6.4 C Search from Context Menu

You can also search for places where the selected element is declared or referenced from the context menu on the editor or [Outline] view.

- (1) Select an element such as a variable or function from the source in the editor or from the [Outline] view and right-click to display the context menu.
- (2) To search for declared locations, select the search range (workspace, current project, or working set) in the [Declarations] submenu.
- (3) To search for referenced locations, select the search range (workspace, current project, or working set) in the [References] submenu.

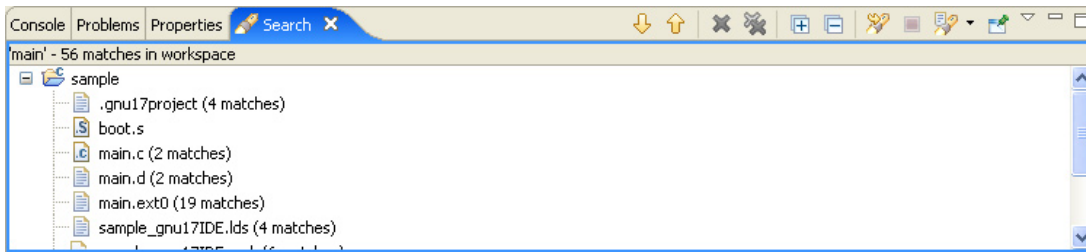
The search results are displayed in the [Search] view.

### 5.6.5 Canceling a Search

The [Search] view is displayed at the start of the search. The [Cancel Current Search] button in the [Search] view toolbar remains enabled while a search is underway. Click this button to cancel the search.

## 5.6.6 Search Results

The results of File and C Searches are displayed in list form in [Search] view.





### Inspecting the search position

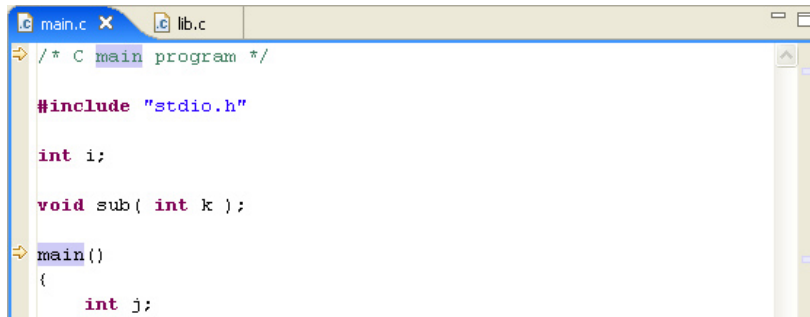
Click one of the search results in the [Search] view to jump to the corresponding location in the editor.

If the target file is not open, double-click in the search results to open it.

You can also use the [Search] view toolbar buttons to navigate the search results.

-  [Show Next Match]      Jumps to the search position immediately following the current search position in the list (equivalent to [Next Annotation] in the [Navigate] menu)
-  [Show Previous Match]      Jumps to the search position immediately preceding the current search position in the list (equivalent to [Previous Annotation] in the [Navigate] menu)

Each occurrence of the search string is shown highlighted in the editor and indicated by an arrow marker in the marker bar for that line.

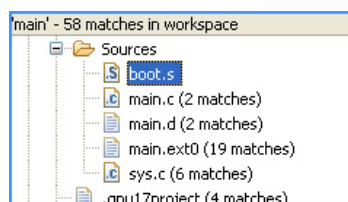


If you left the file name blank for the search, the search begins from the beginning of the file. To find and review a file in the [Navigator] view by file name, select the file name in the [Search] view, then select [Show In] > [Navigator] from the context menu or from the [Navigate] menu. The corresponding file in the [Navigator] view will be highlighted (assuming it is displayed in the list).

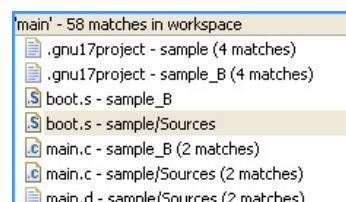
### Changing [Search] view display modes

The [Search] view is initially set to display the search results in tree form. To display the search results in non-hierarchical mode, select [Flat Layout] from the [Show as List] view menu (▽).

Show as Tree

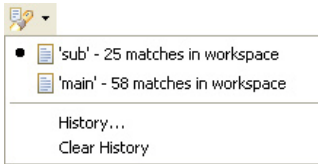


Show as List



## Search history

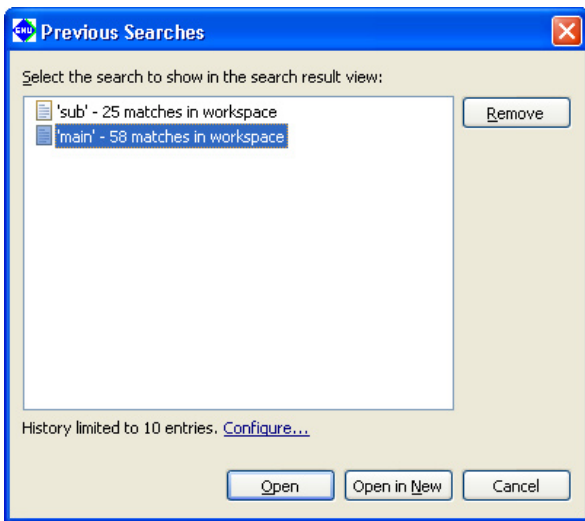
The [Show Previous Searches] shortcut in the [Search] view toolbar displays a list of the file, C, and text searches previously performed.



You can review or repeat a previous search result by selecting the corresponding search from the list.

Select [Clear History] in the [Show Previous Searches] shortcut to delete all previous searches from the history.

When [History...] is selected from the [Show Previous Searches] shortcut, the dialog box below appears to allow you to select the previous searches to be displayed. You can also delete previous searches individually from the history.



### [Remove]

Deletes the previous searches selected from the list.

### [Open]

Displays the results of the previous search selected from the list in the active [Search] view.

### [Open in New]



Opens a new [Search] view and displays the results of the previous search selected from the list.

### [Cancel]

Closes the dialog box.

## Deleting the search results

Use [Search] view toolbar buttons to delete search results.

-  [Remove Selected Matches] Deletes the search results currently selected in the view.
-  [Remove All Matches] Deletes all search results listed in the view.

The search results deleted here will no longer be displayed when you select [Show Previous Searches].

## 5.7 Building a Program

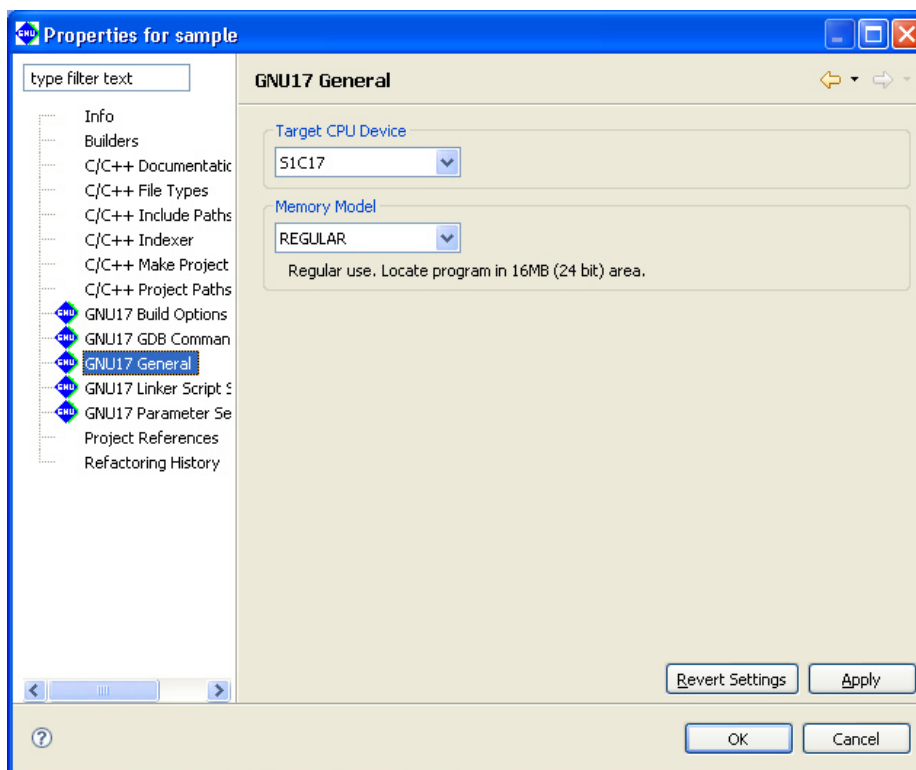
Building a program means compiling/assembling the necessary sources and linking the compiled/assembled sources, including libraries, to generate an executable object file. In practice, this means running **make.exe** to execute the makefile containing the compiler and linker execution procedures.

This section describes how to set the tool options and linker scripts needed for a build operation and how to execute a build process.

### 5.7.1 Setting the CPU Type and Memory Model

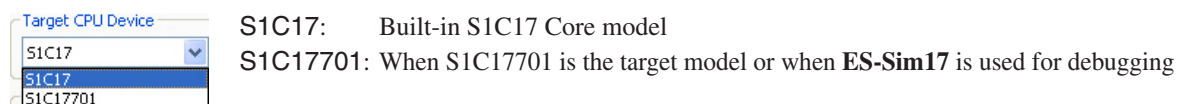
The startup command options for tools and libraries to be linked depend on the processor and its memory space size for which you are developing the application. You must select the correct processor type and memory model before attempting a build process. In most cases, you will not need to select a target CPU type and a memory model, since this would presumably have been done when you created the project. If necessary, you can reset the CPU type and memory model as follows:

- (1) In the [C/C++ Projects] or [Navigator] view, select a project for which you want to change the CPU type or memory model.
- (2) Select [Properties] from the [Project] menu or from the context menu in the above view. This displays the [Properties] dialog box.
- (3) Select [GNU17 General] from the properties list.



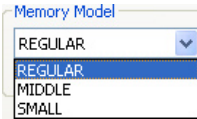
The [Target CPU Device] and [Memory Model] combo boxes show the currently selected C17 Core/processor type and memory model, respectively. (The default CPU type and memory model are S1C17 and REGULAR.)

- (4) From the [Target CPU Device] combo box, select the target processor:



The models displayed in the list may be added/deleted by the configuration file that will be modified when a new model is released or an existing model is discontinued. The selected CPU name will be written in the parameter file to be sent to the debugger and is used for configuration of the debugger's model simulator function. (However, "S1C17" is provided only for core simulation, so it is not described in the parameter file.)

- (5) From the [Memory Model] combo box, select the memory model of the target:



### REGULAR

Address size:

24 bits (16M-byte space can be used)

Compiler and assembler options (described in makefile):

`-mpointer16` Not specified

`-mshort-offset` Not specified

Library files (described in makefile and linker script file):

24-bit libraries

### MIDDLE

Address size:

20 bits (1M-byte space can be used)

Compiler and assembler options (described in makefile):

`-mpointer16` Not specified

`-mshort-offset` Specified

Library files (described in makefile and linker script file):

24-bit libraries

### SMALL

Address size:

16 bits (64K-byte space can be used)

Compiler and assembler options (described in makefile):

`-mpointer16` Specified

`-mshort-offset` Specified

Library files (described in makefile and linker script file):

16-bit libraries

- (6) Click the [OK] button to confirm the changes made or the [Cancel] button to cancel.

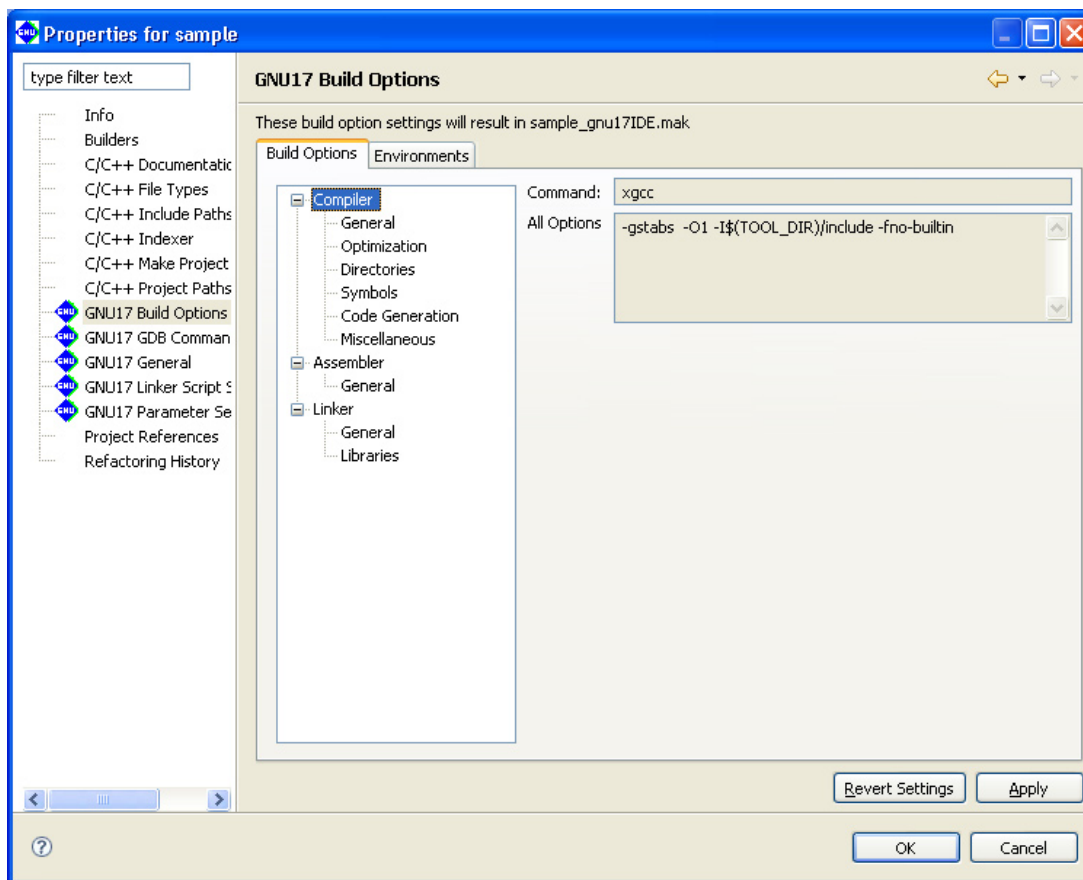
The buttons have the functions described below:

[OK]	Confirms the changes made. If above settings have been changed, a dialog box appears for selecting "clean" build (see Section 5.7.8) to delete the files created with the previous settings (and rebuild). Then the [Properties] dialog box is closed.
[Cancel]	Discards the changes made and closes the dialog box.
[Apply]	Confirms the changes made, but will not close the dialog box. To change other properties, click the [Apply] button before proceeding to the desired page. If above settings have been changed, a dialog box appears for selecting "clean" build (see Section 5.7.8) to delete the files created with the previous settings (and rebuild).
[Revert Settings]	Undoes the changes made, restoring the state in which this page was opened (or, if you clicked the [Apply] button, the content confirmed at that point).

## 5.7.2 Setting Compiler Options

Do the following to set C compiler command options.

- (1) Select a project to build in the [C/C++ Projects] or [Navigator] view.
- (2) Select [Properties] from the [Project] menu, or select the context menu from the above view.  
This displays the [Properties] dialog box.
- (3) Select [GNU17 Build Options] from the properties list.
- (4) Select [Compiler] from the [Build Options] tree.



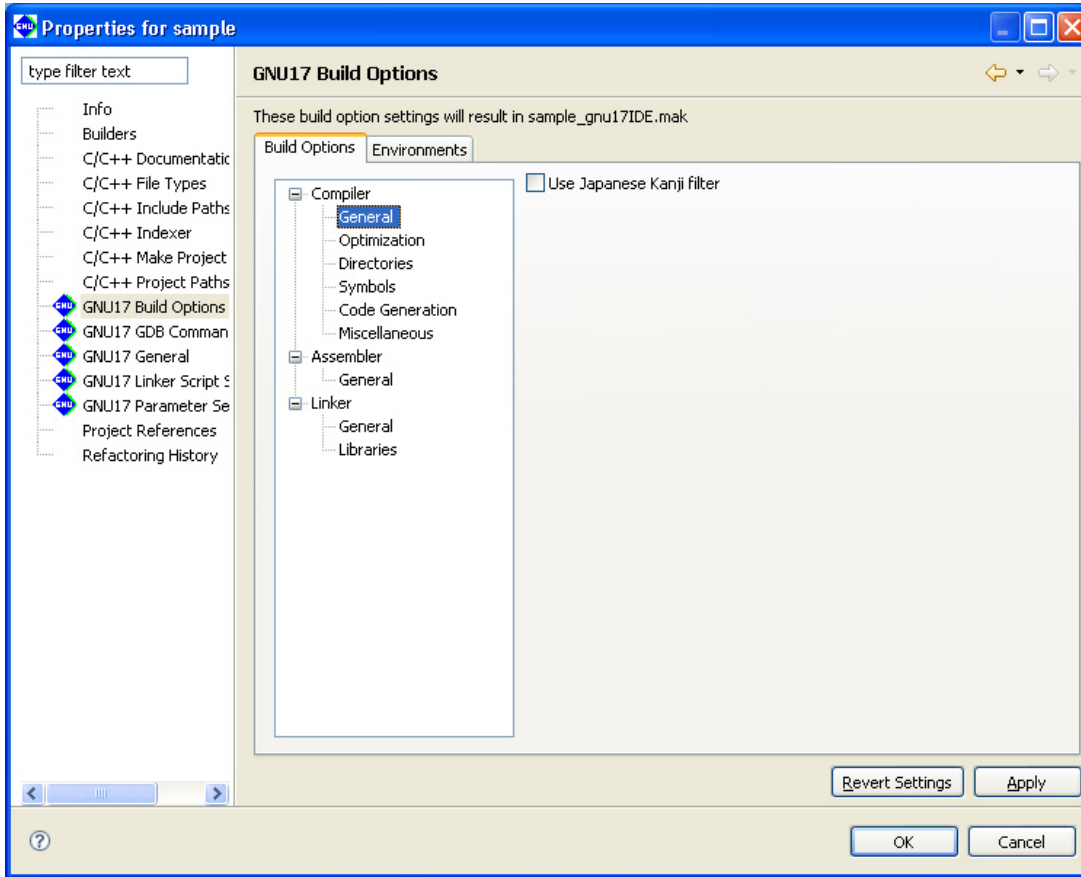
The [Command:] field shows the name of the C compiler. The [All Options] field lists currently set compiler options.

- (5) Select a category from the [Compiler] tree list and set the necessary options.
- (6) Click the [Apply] button to change other properties or the [OK] button to complete property settings.  
If settings in a [GNU17 Build Options] page have been changed, a dialog box appears for selecting "clean" build (see Section 5.7.8) to delete the files created with the previous settings (and rebuild).

If you haven't clicked [Apply] yet, you can use the [Revert Settings] button to discard the changes and restore the state in which this page was opened.

Shown below are the pages in which compiler options are set for each category. For detailed information on the options, refer to the section that discusses the C compiler.

## [General]



Select basic compiler option from this page.

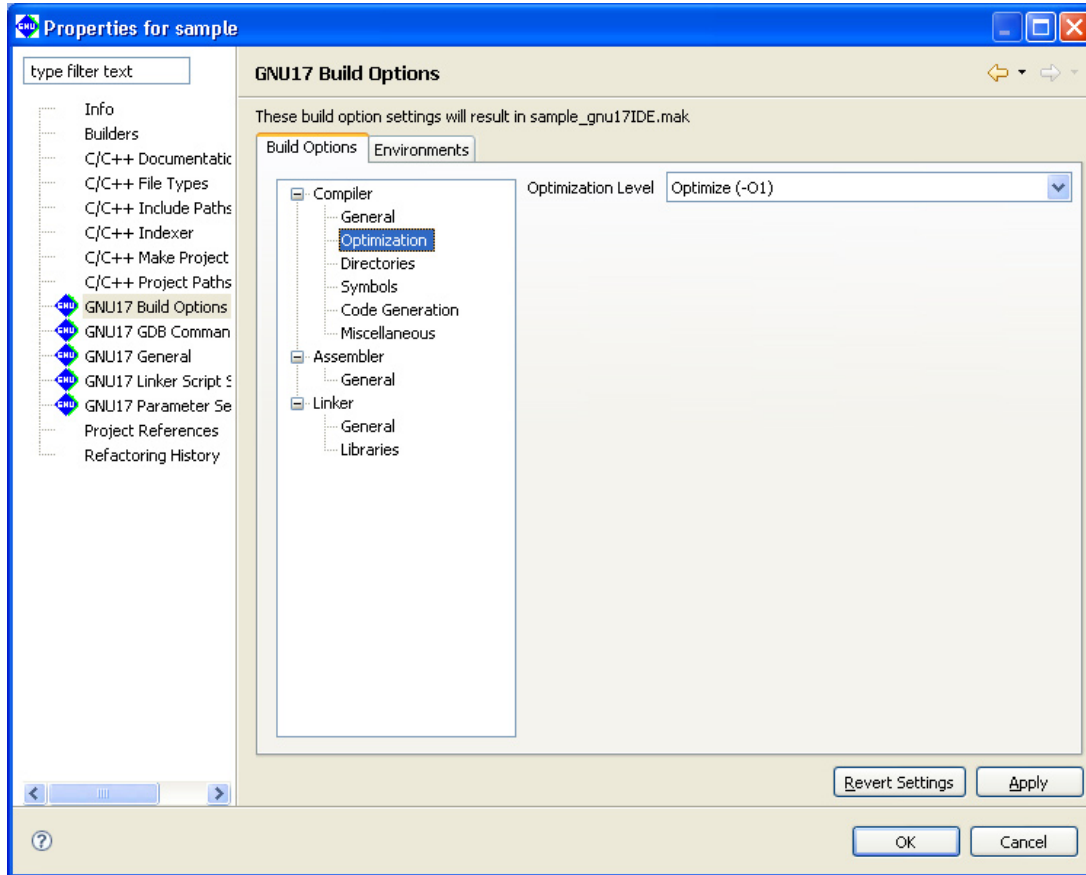
## [Use Japanese Kanji filter]

If this option is specified, the compiler converts occurrences of the Shift JIS code in the source into the ASCII escape character before calling the compiler.

The default check box status depends on the OS under which the **IDE** starts up; the check box is set to on under an OS with Japanese environment or is set to off under other language versions.



## [Optimization]

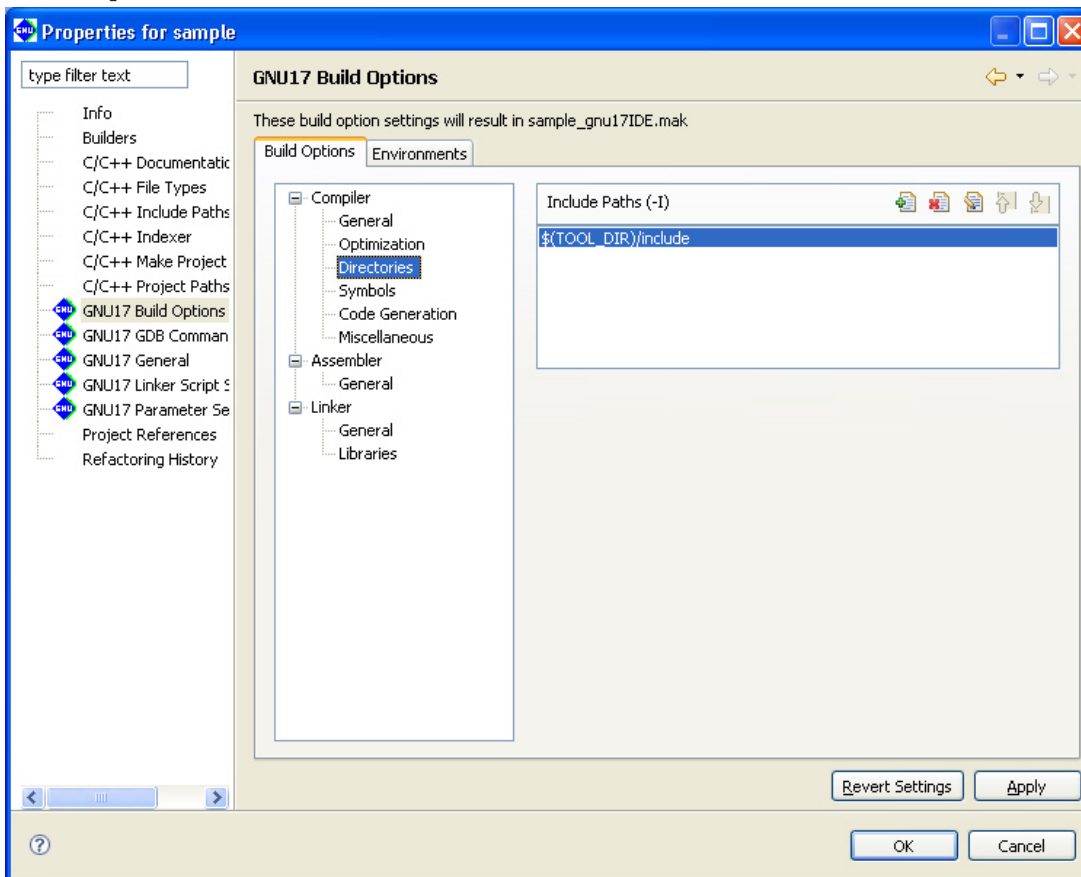


Select compiler optimization options from this page.

[Optimization Level] (default: -O1)

Select an optimization level (-O1).






## [Directories]



Set compiler search path options from this page.

[Include Paths (-I)] (default: `-I$(TOOL_DIR)/include`)

Set the include file search path. The buttons have the functions described below.

-  [Add] Adds a directory. A dialog box for entering a path or selecting one using the [Browse...] button is displayed.
-  [Delete] Deletes the path selected in the list.
-  [Edit] Edits the path selected in the list. A dialog box is displayed to allow you to edit the path.
-  [Move Up] Moves the path selected in the list one position up in the list. The include files are searched in order in which the paths are listed, beginning with the uppermost path.
-  [Move Down] Moves the path selected in the list one position down.

\* About `$(TOOL_DIR)`

The [Include Paths (-I)] column lists "`$(TOOL_DIR)/include`" that is set by default.

`$(environment variable)` is a macro defined in the makefile that is generated when you build a project. `TOOL_DIR` is the environment variable in which the path to the gnu17 tool directory is defined. The defined contents can be verified in the [Environments] tab page.

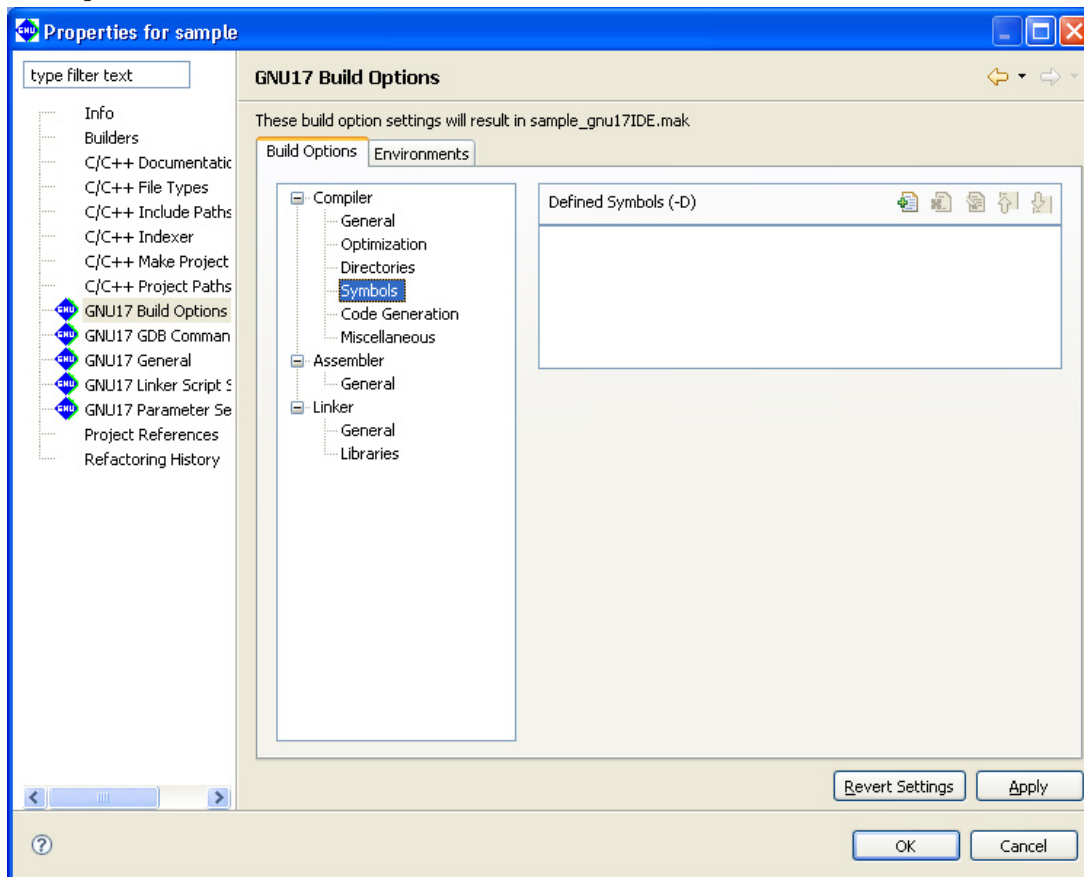
Example: If the gnu17 tools have been installed in the `c:\EPSON\gnu17` directory

```
TOOL_DIR = c:/EPSON/gnu17
```

Since the macro is replaced with the contents of the environment variable described in ( ) during execution of **make.exe**, `-I$(TOOL_DIR)/include` will be resolved to `-Ic:/EPSON/gnu17/include`.

The [Environments] tab page allows the user to define environment variables similar to `TOOL_DIR`. The environment variables defined here may be used for specifying include file and library file paths in the build options. Refer to Section 5.10.1 for details of the [Environments] page.


## [Symbols]



Set compiler macro-definition options from this page.





## [Defined Symbols (-D)] (default: none)

Specify a macro-name and replacement character. The buttons have the functions described below.

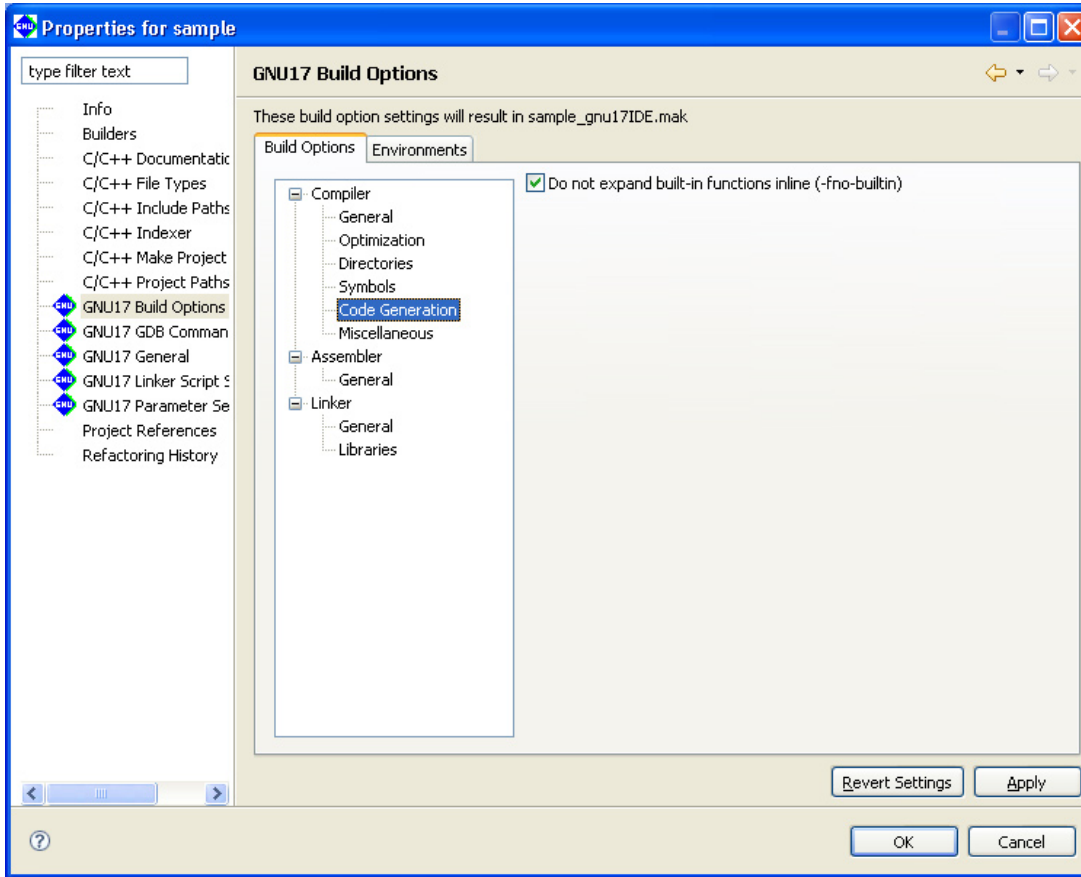
-  [Add] Adds a macro-definition. A dialog box for entering a macro-definition is displayed. Make the entry in the form shown below.  

$$\langle \text{macro-name} \rangle$$

or

$$\langle \text{macro-name} \rangle = \langle \text{replacement string} \rangle$$
-  [Delete] Deletes the selected macro-definition from the list.
-  [Edit] Edits the macro-definition selected in the list. A dialog box is displayed to allow you to edit the macro-definition.
-  [Move Up] Moves the macro-definition selected in the list one position up in the list.
-  [Move Down] Moves the macro-definition selected in the list one position down.

## [Code Generation]

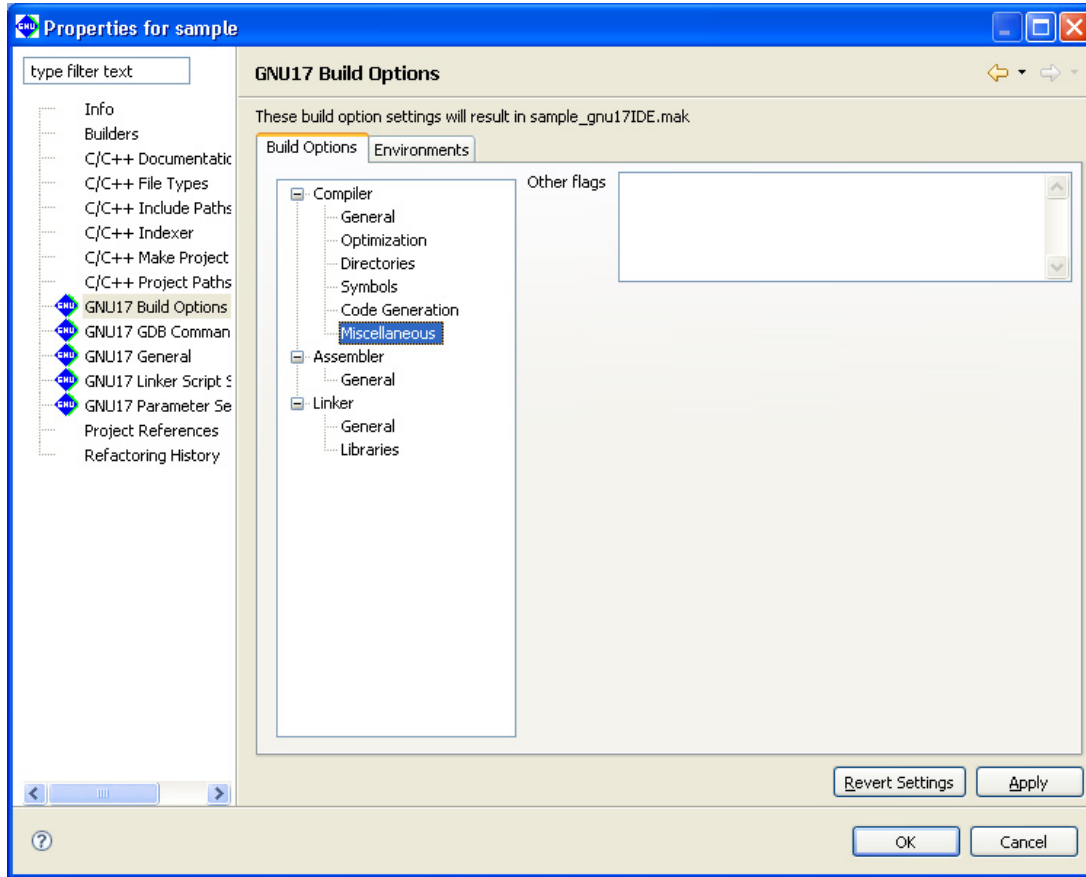


Select compiler code generation linker option from this page.

[Do not expand built-in functions inline (-fno-builtin)] (default: ON)

If this option is specified, built-in functions are ignored and the functions are always called.

For the functions in question, refer to Section 6.3.2, "Command-line Options".

**[Miscellaneous]**

Set other compiler options from this page.

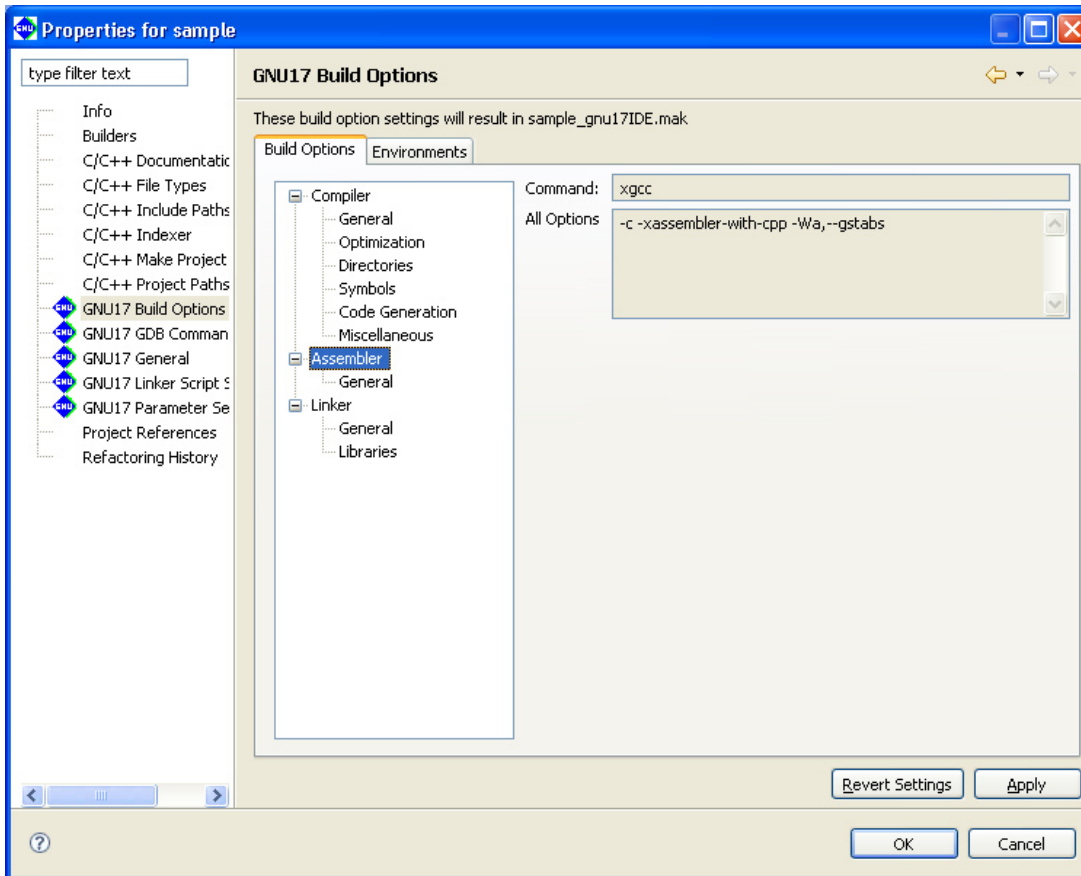
**[Other flags]** (default: none)

Enter other options directly into this text field. Separate each option with one or more spaces.

### 5.7.3 Setting Assembler Options

Do the following to set assembler command options.

- (1) Select a project to build in the [C/C++ Projects] or [Navigator] view.
- (2) Select [Properties] from the [Project] menu, or select the context menu from the above view.  
This displays the [Properties] dialog box.
- (3) Select [GNU17 Build Options] from the properties list.
- (4) Select [Assembler] from the [Build Options] tree.



The [Command:] field shows the program name of the compiler\*, and the [All Options] field lists the currently set options.

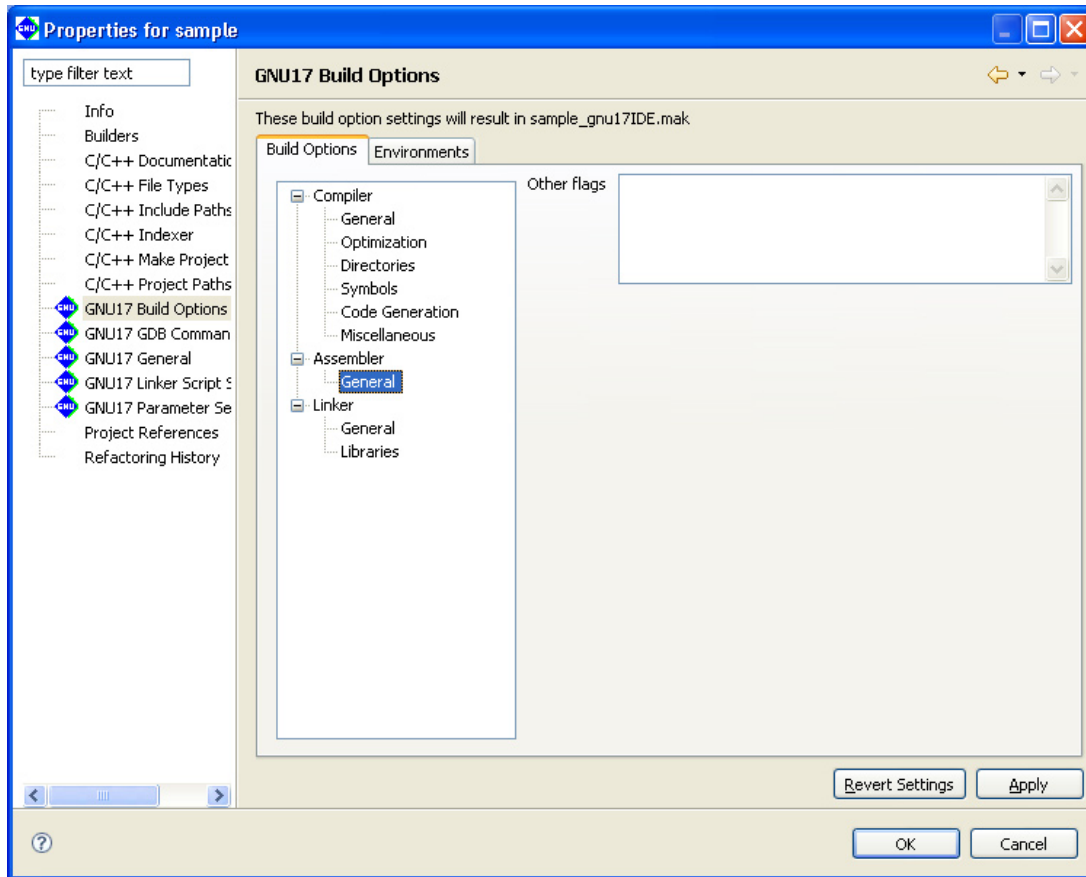
\* In the IDE, assembler sources are assembled by the C compiler with the `-xassembler-with-cpp` option specified.

- (5) Select [General] from the [Assembler] tree list and set the necessary options.
- (6) Click the [Apply] button to change other properties or the [OK] button to complete property settings.  
If settings in a [GNU17 Build Options] page have been changed, a dialog box appears for selecting "clean" build (see Section 5.7.8) to delete the files created with the previous settings (and rebuild).

If you haven't clicked [Apply], you can use the [Revert Settings] button to discard the changes made and restore the state in which this page was opened.

Shown below are the pages in which assembler options are set. For detailed information on the options, refer to the section that discusses the assembler.

## [General]



The `-c`, `-xassembler-with-cpp`, and `-Wa, --gstabs` options (and `-mpointer16` option depending on the memory model selected) are always added. Set other assembler options from this page.

## [Other flags] (default: none)

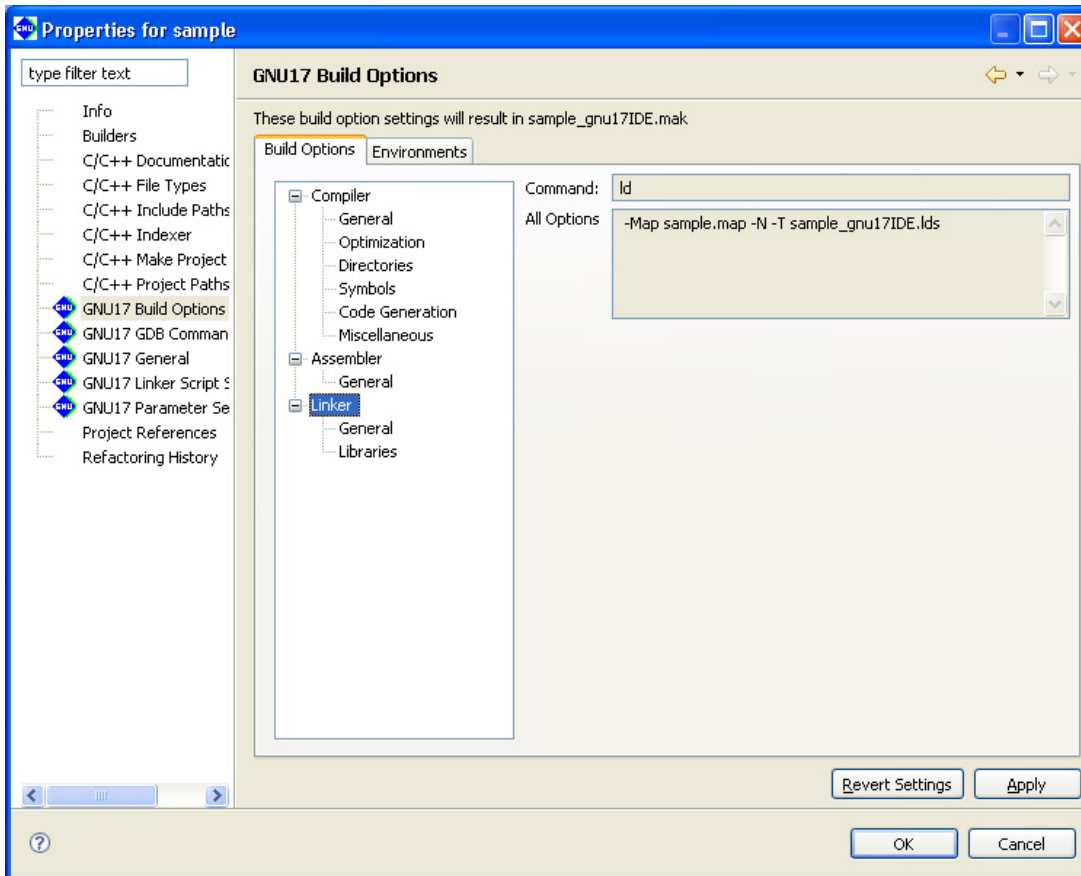
Enter the options to be passed to the assembler. Insert one or more spaces between each option.

The options entered are passed to the assembler as `"-Wa, <option>, ..."`.

## 5.7.4 Setting Linker Options

Do the following to set the linker command options:

- (1) Select a project to build in the [C/C++ Projects] or [Navigator] view.
- (2) Select [Properties] from the [Project] menu, or select the context menu from the above view.  
This displays the [Properties] dialog box.
- (3) Select [GNU17 Build Options] from the properties list.
- (4) Select [Linker] from the [Build Options] tree.



The [Command:] field shows the program name of the linker. The [All Options] field lists the currently set options.

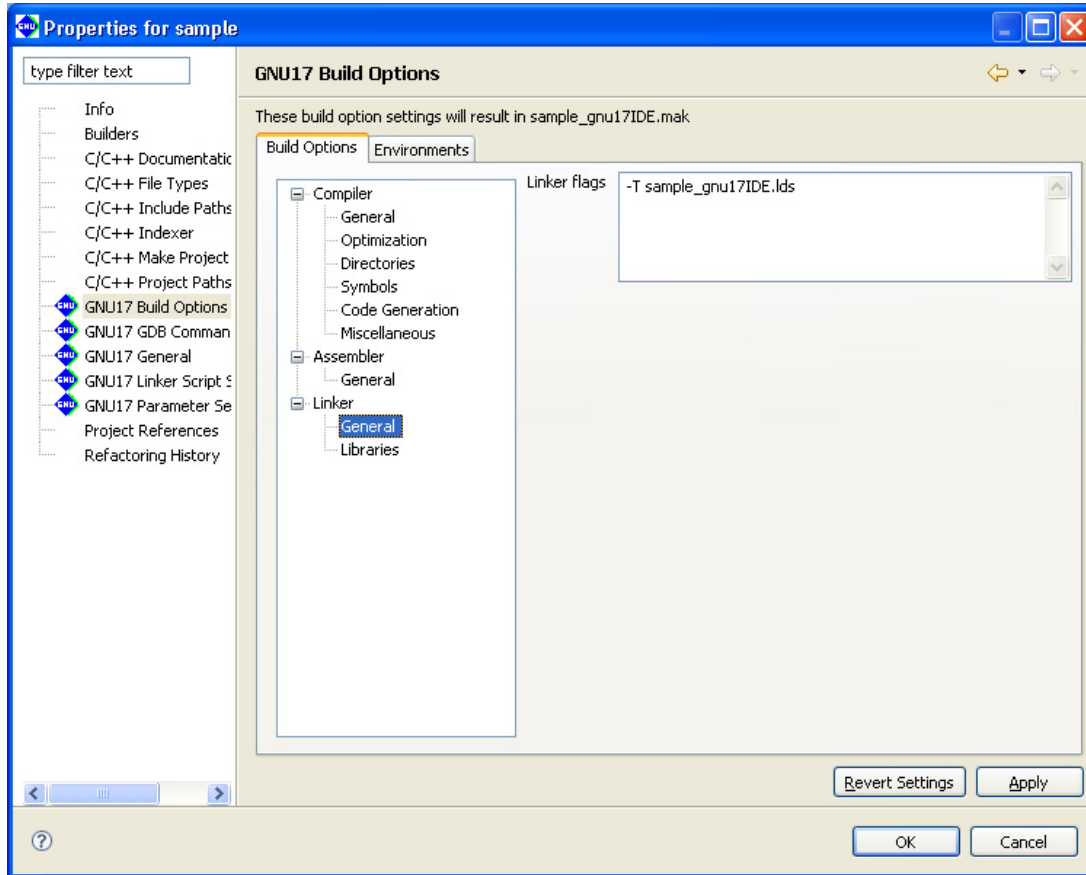
- (5) Select a category from the [Linker] tree list and set the necessary options.
- (6) Click the [Apply] button to change other properties or the [OK] button to complete property settings.  
If settings in a [GNU17 Build Options] page have been changed, a dialog box appears for selecting "clean" build (see Section 5.7.8) to delete the files created with the previous settings (and rebuild).

If you haven't clicked [Apply], you can use the [Revert Settings] button to discard the changes made and to return to the state in which this page was opened.

Shown below are the pages in which the linker options are set. For detailed information on the options, refer to the section that discusses the linker.



## [General]

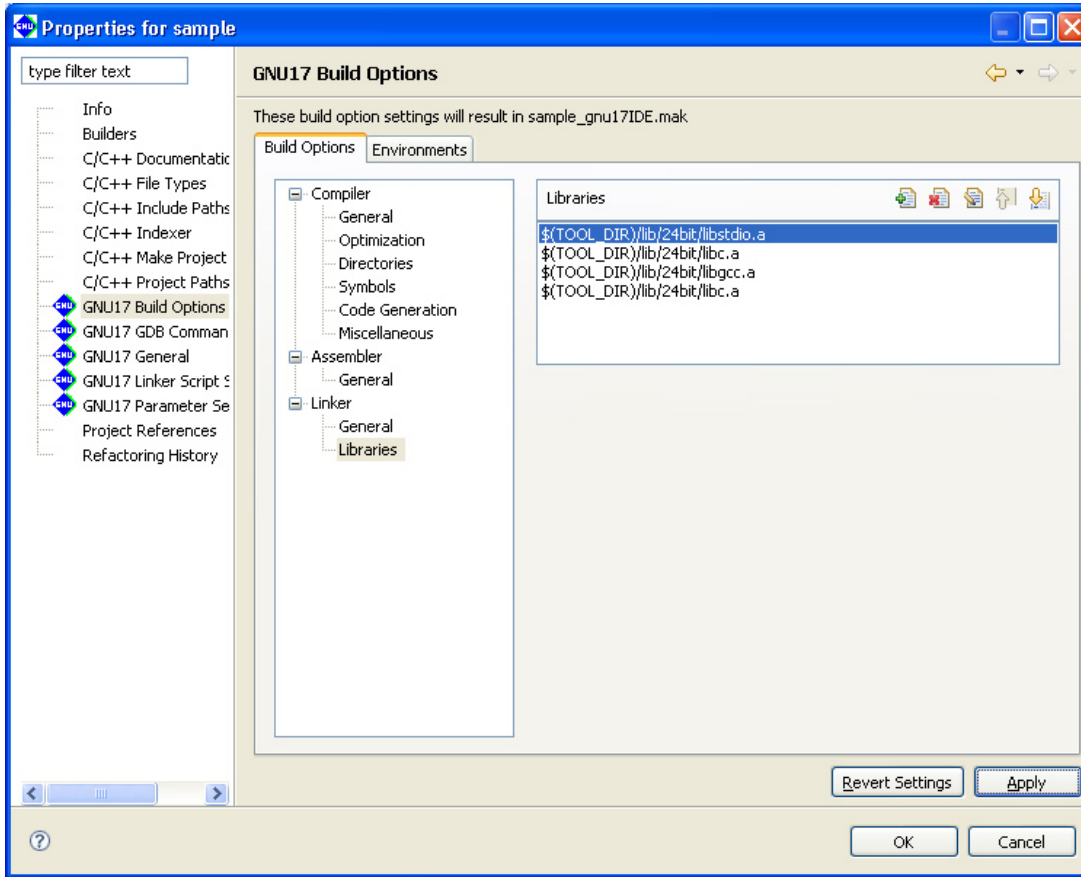


The `-Map` and `-N` options are always added. Set other linker options from this page.

[Linker flags] (default: `-T<project name>_gnu17IDE.lds`)

Enter other linker options in this text field. Insert one or more spaces between each option.






## [Libraries]



Set the libraries to be linked from this page.

[Libraries] (default: libstdc.a, libc.a, libgcc.a, libc.a\*)

Set the libraries to be linked. The buttons have the functions described below.

-  [Add] Adds a library. A dialog box is displayed to allow you to enter a path or select one using the [Browse...] button.
-  [Delete] Deletes the library selected in the list.
-  [Edit] Edits the library selected in the list. A dialog box is displayed to allow you to edit the path.
-  [Move Up] Moves the library selected in the list one position up in the list. Libraries are linked in order of listed paths, beginning with the uppermost path.
-  [Move Down] Moves the library selected in the list one position down.

The libraries set here are written in a makefile to link to the objects generated from the sources. However, they must be mapped to sections in a linker script file.

- \* Either the 24-bit libraries or 16-bit libraries are specified by default according to the selected memory model. Furthermore, libc.a is specified twice to resolve cross-references between libc.a and libgcc.a.

\* About \$(TOOL\_DIR)

The [Libraries] column lists "\$(TOOL\_DIR)/lib/24bit(16bit)/libxxx.a" that is set by default. \$(*environment variable*) is a macro defined in the makefile that is generated when you build a project. TOOL\_DIR is the environment variable in which the path to the gnu17 tool directory is defined. The defined contents can be verified in the [Environments] tab page.

Example: If the gnu17 tools have been installed in the c:\EPSON\gnu17 directory

```
TOOL_DIR = c:/EPSON/gnu17
```

Since the macro is replaced with the contents of the environment variable described in ( ) during execution of **make.exe**, -I\$(TOOL\_DIR)/lib/24bit/libxxx.a will be resolved to -Ic:/EPSON/gnu17/lib/24bit/libxxx.a.

The [Environments] tab page allows the user to define environment variables similar to TOOL\_DIR. The environment variables defined here may be used for specifying include file and library file paths in the build options. Refer to Section 5.10.1 for details of the [Environments] page.

## 5.7.5 Generated Makefile

Building a project generates a makefile named "<project name>\_gnu17IDE.mak" according to the CPU type and the tool options set above, which is then executed by **make.exe**.

An example of a generated makefile is shown below.

Example:

```
# Make file generated by Gnu17 Plug-in for Eclipse
# This file should be placed directly under the project folder

# macro definitions for target file (1)
TARGET= sample

# macro definitions for tools (2)
TOOL_DIR= C:/EPSON/gnu17
CC= $(TOOL_DIR)/xgcc
AS= $(TOOL_DIR)/xgcc
AS_CC= $(TOOL_DIR)/as
LD= $(TOOL_DIR)/ld
RM= $(TOOL_DIR)/rm
SED= $(TOOL_DIR)/sed
CC_KFILT= $(TOOL_DIR)/xgcc_filt
OBJDUMP= $(TOOL_DIR)/objdump

# macro definitions for tool flags (3)
CFLAGS= -B$(TOOL_DIR)/ -gstabs -S -O1 -I$(TOOL_DIR)/include -fno-builtin
ASFLAGS= -B$(TOOL_DIR)/ -c -xassembler-with-cpp -Wa,--gstabs
ASFLAGS_CC=
LDFLAGS= -Map sample.map -N -T sample_gnu17IDE.lds
EXTFLAGS= -Wa,-mc17_ext -Wa,$(TARGET).dump -Wa,$(TARGET).map
EXTFLAGS_CC= -mc17_ext $(TARGET).dump $(TARGET).map
OBJDUMPFLAGS= -t

# macro definitions for object files (4)
OBJS= boot.o \
      lib.o \
      main.o \
      sys.o \

# macro definitions for library files (5)
OBJLDS= $(TOOL_DIR)/lib/24bit/libstdio.a \
        $(TOOL_DIR)/lib/24bit/libc.a \
        $(TOOL_DIR)/lib/24bit/libgcc.a \
        $(TOOL_DIR)/lib/24bit/libc.a \

# macro definitions for assembly files generated from c source files (6)
CEXTTEMPS= lib.ext0 \
           main.ext0 \
           sys.ext0 \

# macro definitions for original assembly files
ASMEXTTEMPS= boot.s \

# macro definitions for dependency files (7)
DEPS= $(OBJS:%.o=%.d)
SED_PTN= 's/[[:space:]]\([a-zA-Z]\)\:/ \cygdrive\\1/g'
SED_PTN2= 's/^\($subst .,\.,$(@F)\)\:/$(subst /,/,,$(@))\:/g'

# macro definitions for creating dependency files (8)
DEPCMD_CC= @$ (CC) -M -MG $(CFLAGS) $< | $(SED) -e $(SED_PTN) | $(SED) -e $(SED_PTN2)
>$ (@:%.o=%.d)
DEPCMD_AS= @$ (AS) -M -MG $(ASFLAGS) $< | $(SED) -e $(SED_PTN) | $(SED) -e
$(SED_PTN2) >$ (@:%.o=%.d)
```

```
# targets and dependencies (9)
```

```
.PHONY : all clean
```

```
all : $(TARGET).elf (10)
```

```
$(TARGET).elf : $(OBJS) sample_gnu17IDE.mak sample_gnu17IDE.lds
```

```
# 1pass linking
```

```
$(LD) $(LDFLAGS) -o $@ $(OBJS) $(OBJLDS)
```

```
$(OBJDUMP) $(OBJDUMPFLAGS) $@ > $(TARGET).dump
```

```
$(RM) -f $(OBJS) $(TARGET).elf
```

```
# 2pass for assembly files
```

```
for NAME in $(basename $(ASMEXTTEMPS)) ; do \
```

```
$(AS) $(ASFLAGS) $(EXTFLAGS) -o $$NAME.o $$NAME.s ; done
```

```
# 2pass for c files
```

```
for NAME in $(basename $(CEXTTEMPS)) ; do \
```

```
$(AS_CC) $(ASFLAGS_CC) $(EXTFLAGS_CC) -o $$NAME.o $$NAME.ext0 ; done
```

```
$(RM) -f $(TARGET).map
```

```
# 2pass linking
```

```
$(LD) $(LDFLAGS) -o $@ $(OBJS) $(OBJLDS)
```

```
## boot.s (11)
```

```
boot.o : boot.s
```

```
$(AS) $(ASFLAGS) -o $@ $<
```

```
$(DEPCMD_AS)
```

```
## lib.c
```

```
lib.o : lib.c lib.ext0
```

```
$(CC) $(CFLAGS) -o $@ $@.o $@.ext0 $<
```

```
$(AS_CC) $(ASFLAGS_CC) -o $@ $@.o $@.ext0
```

```
$(DEPCMD_CC)
```

```
## main.c
```

```
main.o : main.c main.ext0
```

```
$(CC) $(CFLAGS) -o $@ $@.o $@.ext0 $<
```

```
$(AS_CC) $(ASFLAGS_CC) -o $@ $@.o $@.ext0
```

```
$(DEPCMD_CC)
```

```
## sys.c
```

```
sys.o : sys.c sys.ext0
```

```
$(CC) $(CFLAGS) -o $@ $@.o $@.ext0 $<
```

```
$(AS_CC) $(ASFLAGS_CC) -o $@ $@.o $@.ext0
```

```
$(DEPCMD_CC)
```

```
# dependencies for assembled c source files
```

```
$(CEXTTEMPS) : $(CEXTTEMPS:%.ext0=%.c)
```

```
# include dependency files
```

```
-include $(DEPS)
```

```
# clean files (12)
```

```
clean :
```

```
$(RM) -f $(OBJS) $(TARGET).elf $(TARGET).map $(DEPS) $(CEXTTEMPS) $(TARGET).dump
```

Each field indicated by a number is described below.

- (1) Defines the project name as TARGET. This name is used in the elf object file and map file.
- (2) Defines the tool directory and the compiler, assembler, and linker commands. The directory in which the tools are stored is set in TOOL\_DIR. The space characters in the path are converted to "\ " (\ + space). The link process cannot proceed if the path includes spaces. Confirm that one set of S5U1C17001C tools is installed in a directory that does not include spaces.
- (3) Defines the compiler, assembler, and linker options. These options reflect the selected contents of project properties ([GNU17 Build Options]).
- (4) The object file names corresponding to the source files in the project are written here following "OBS=". The contents written in this field change when source files are added or deleted.
- (5) The library file names set in [GNU17 Build Options] > [Build Options] > [Linker] > [Libraries] are written here following "OBLDS=" in the same way as for "OBS=".
- (6) The assembler source files are written here for a two-pass make that optimizes extended instructions. Note that the assembler source files created from C source files have a file extension ".ext0".
- (7) Defines the macros needed to create dependency files. Dependency files are generated for each source. The sources and include files needed to generate object files are defined here.

Example:

```

Dependency file (main.d)
  main.o: main.c

Dependency file (boot.d)
  boot.o: boot.s

```

These files are used to create the tool commands written to a dependency list.

- (8) Defines the execution commands to be stored in a dependency list.
- (9) Defines the target.
- (10) This is the dependency list for the target to be built and executable format object files. The two-pass make process generates the executable format object file after optimizing the extended instructions.
- (11) This is the dependency list for object files generated from each source. Adding or deleting source files will change the information in this field.
- (12) The commands written here delete the generated files executed in the target "clean".

For detailed information on makefiles, refer to Section 11.1, "make.exe".

## 5.7.6 Editing a Linker Script

A linker script file is used to pass location information on object files comprising the execution file (.elf) to the linker. The IDE generates a linker script file "*<project name>\_gnu17IDE.lds*" according to the settings discussed in this section.

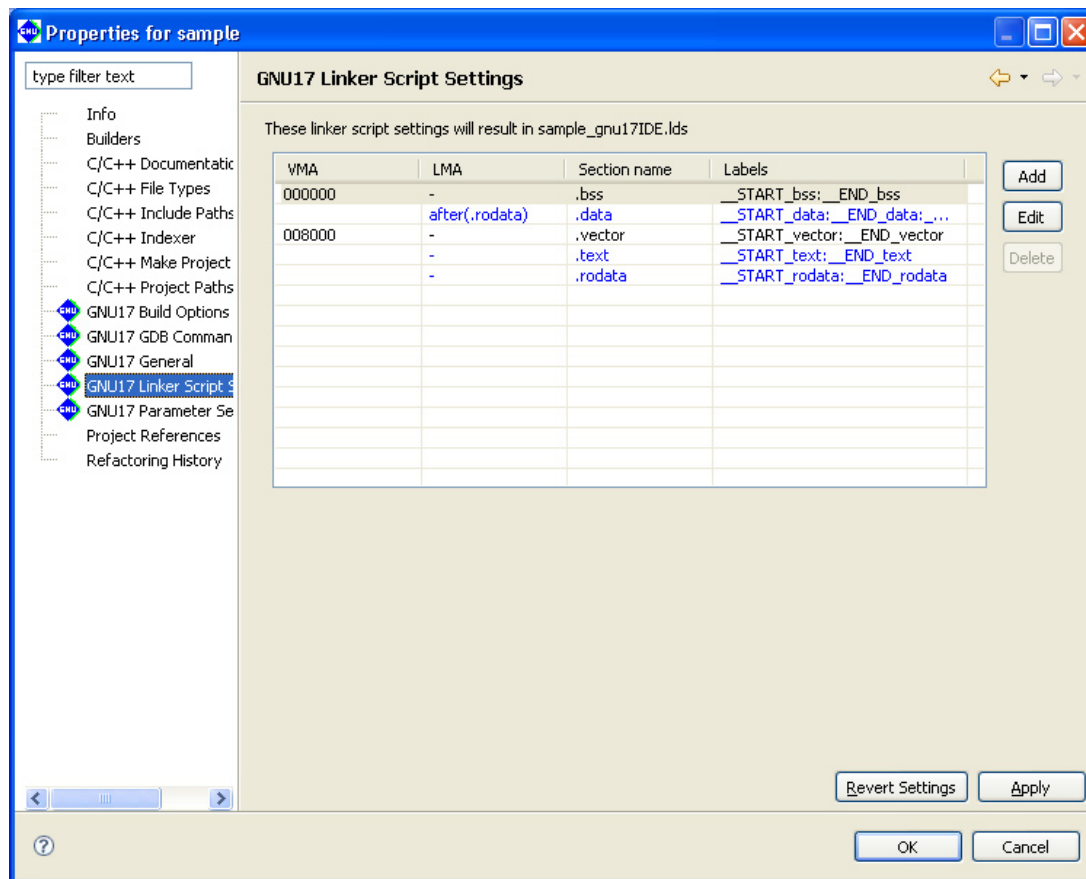
The procedure for setting a linker script is described below. For detailed information on sections and linker scripts, refer to Section 3.8, "Sections and Linkage", and Chapter 9, "Linker".

### Linker script setup page

Use the [GNU17 Linker Script Settings] page of project properties to set a linker script.

Do the following to display the setup page:

- (1) Select a project to build in the [C/C++ Projects] or [Navigator] view.
- (2) Select [Properties] from the [Project] menu or from the context menu in the above view.  
This displays the [Properties] dialog box.
- (3) Select [GNU17 Linker Script Settings] from the properties list.



The list shows the configuration and location of sections in an execution file (.elf). This memory map is shown in Figure 5.7.6.1.

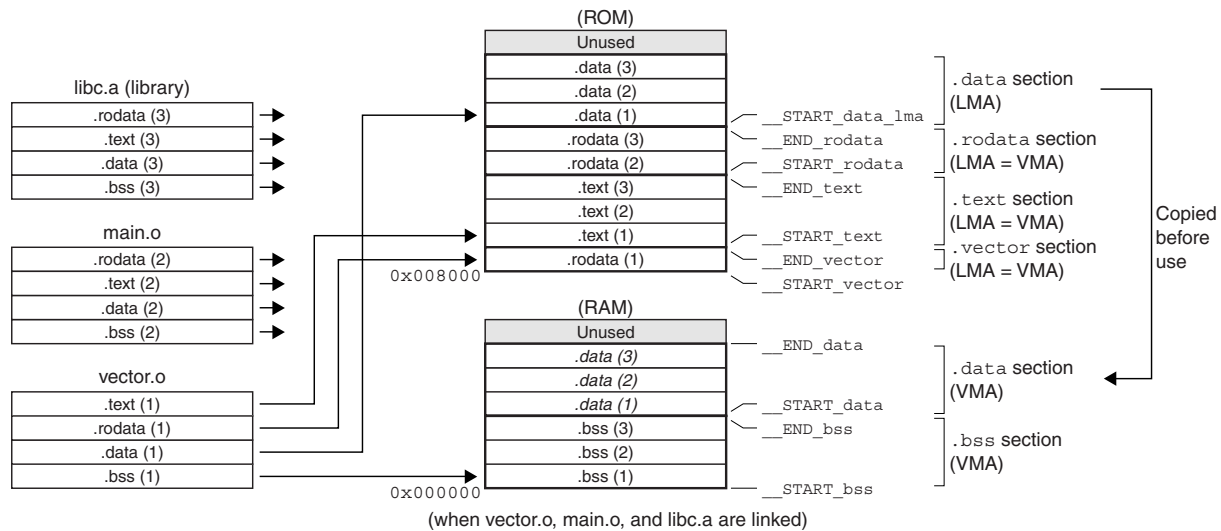


Figure 5.7.6.1 Selection location in default settings

As shown in the [Section name] column, the following five basic sections are set in advance:

- `.bss`: A section in which variables without initial values are placed. (This is normally located in RAM.)
- `.data`: A section in which variables with initial values are placed. (The initial values are located in ROM. When needed, they are copied into RAM.)
- `.vector`: A section in which vector tables are placed. (The actual data is located in ROM.)
- `.text`: A section in which program code is placed. (The actual data is located in ROM and executed from there or from high-speed RAM after copying.)
- `.rodata`: Constants. (The actual data is located in ROM.)

The section information is displayed in blue except for the `.vector` section displayed in black. Blue is used to display the standard sections defined by default and black is used to display other user defined sections. To edit the section name, standard section attribute, address to locate, and objects to be located, a user section should be created. The standard section allows the user to specify the location address only, and objects are automatically located except those are located in the user sections with the same attribute.

The "VMA" (Virtual Memory Address) is the position (start address) at which a section is placed when executed. A section whose address is not written in the VMA will be located at an address following the section immediately preceding.

The "LMA" (Load Memory Address) is the position in a ROM (start address) at which the actual data is located. "-" means the same as the VMA (i.e., a section will be executed or accessed from the position at which its actual data is placed). "after (.rodata)" means that a section will have its actual data located at an address following another section (in this case, the `.rodata` section).

"Labels" are the labels indicating the start and the end addresses of the area in which a section will be located. When a VMA is specified, two labels are displayed, whereas when a LMA is specified, four labels for the start/end VMA addresses and the start/end LMA addresses are displayed, in that order. These labels can be used to specify the address in a source file when (for example) a section is copied from ROM to RAM. The names of these labels are automatically generated from section names.

Example:

```
__START_bss: __END_bss
```

Labels indicating the start and end addresses of a `.bss` section

```
__START_data: __END_data: __START_data_lma: __END_data_lma
```

Labels indicating the start VMA address, end VMA address, start LMA address, and end LMA address of a `.data` section



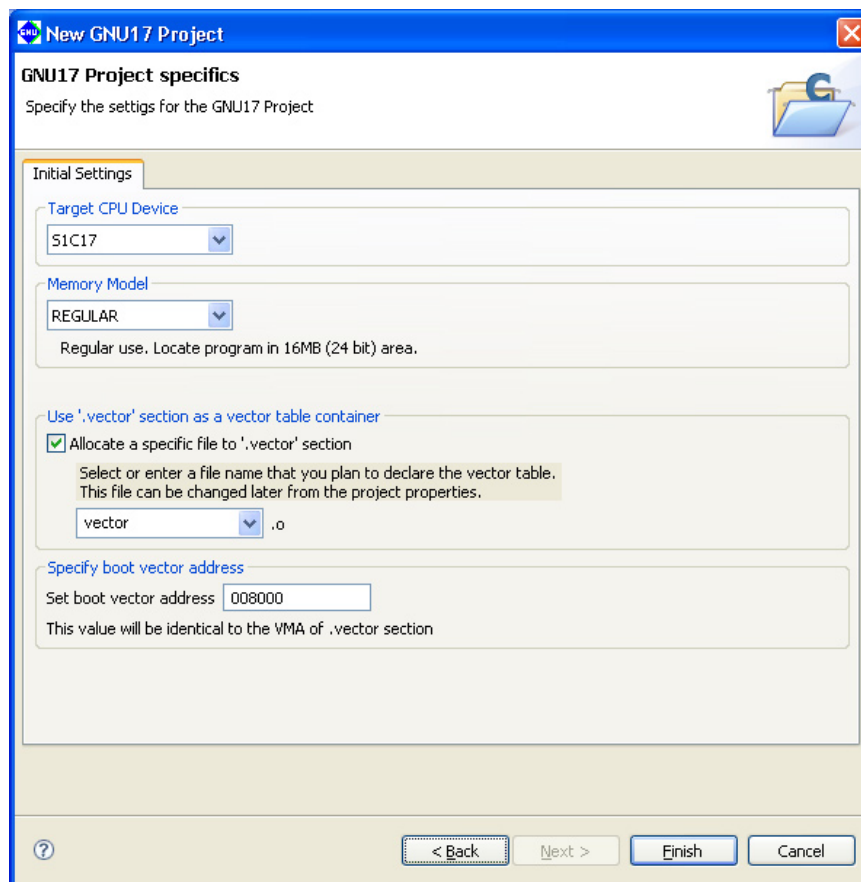
## About the `.vector` section

The `.vector` section is an exclusive section provided by the **IDE** to ensure that vector tables will always be located beginning with the trap table vector address.

With the initial **IDE** settings, the `.text` section is located immediately after the `.vector` section.

By specifying a file that includes a vector table as the object to be located in this section, it is possible to ensure, without concern for the order in which this and other objects are located, that the vector table will always be located from the above address.

The object to be located in this section can be selected in the wizard for creating a new project. For more information on this wizard, refer to Section 5.4.2, "Creating a New Project".



Select the object you want to locate in the `.vector` section from the combo box list (`vector.o` and `boot.o` selectable) or by entering it in the combo box text field.

Use the [Set boot vector address] text box to specify the address to locate the `.vector` section. The default value is "008000". The value set here will be used as the parameter for the `TTBR` setting command that will be written in the debugger startup command file created by the **IDE** as well as it will be used as the VMA of the `.vector` section that will be written in the linker script file.

If you do not locate objects in the `.vector` section, deselect the [Allocate a specific file to `'.vector'` section] check box. Even so, the `.vector` section is defined as a section, but without an object.

The contents set in the wizard can be changed in the [Edit Section] dialog box (refer to "Editing section information").

The `.vector` section is defined with the `.rodata` attribute. Make sure the vector table is written in the source files, as shown below.

#### For C sources (`vector.c`)

Declare a vector table with `const` to specify that it be located in the `.rodata` section.

Example:

```
const unsigned long vector[] = {
    (unsigned long)boot,           // 0x0    0
    (unsigned long)addr_err,      // 0x4    1
    (unsigned long)nmi,          // 0x8    2
                                :
    (unsigned long)dummy,        // 0x48   18
    (unsigned long)dummy        // 0x4c   19
};
```

#### For assembler sources (`boot.s`)

Declare a `.rodata` section and write a vector table following it.

Example:

```
.section .rodata, "a"
.long   BOOT           ; 0x0    0
.long   ADDR_ERR      ; 0x4    1
.long   NMI           ; 0x8    2
                                :
.long   DUMMY         ; 0x48   18
.long   DUMMY         ; 0x4c   19
```

If you are using an assembler source in which a vector table is written in the `.text` section and you want the table to be located in the `.vector` section, select the desired method from the following options:

#### Method 1: Editing the source file

- (1) Insert a `.rodata` directive similar to the one shown above before the vector table in the source file. If a program is written after the vector table, insert a `.text` directive in front of it and declare a `.text` section.
- (2) In the new project wizard, select the [Allocate a specific file to '.vector' section] check box, then `boot.o` in the combo box. (If the source is other than `boot.s`, enter the file name of the source.)

#### Method 2: Editing section information in the IDE (using the source file as is)

- (1) In the new project wizard, select the [Allocate a specific file to '.vector' section] check box, then `boot.o` in the combo box. (If the source is other than `boot.s`, enter the file name of the source.)
- (2) In the [Edit Section] dialog box, change the attribute of the `.vector` section to `.text`. (Refer to the discussion in the next and the following pages.)

If you are not using the `.vector` section, deselect the [Allocate a specific file to '.vector' section] check box and edit the `.text` section in the [Edit Section] dialog box to locate `boot.o` at the top.

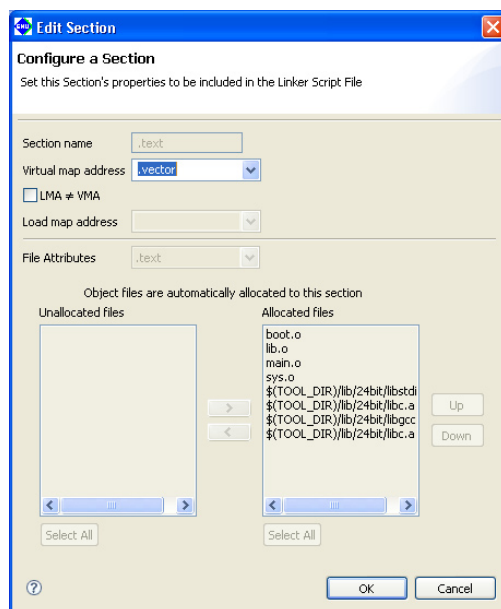
## Editing section information

The location information on each section described above and the objects or libraries to be located in the respective sections can be changed as suitable for the system. The procedure is described below.

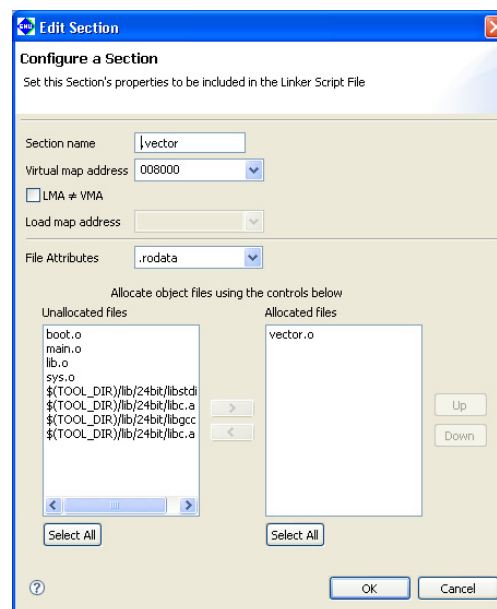
- (1) Click the section you want to edit in the section list for the [GNU17 Linker Script Settings] page.
- (2) Click the [Edit] button.  
This displays the [Edit Section] dialog box.
- (3) Make the necessary changes according to the explanation given below. Click [OK].
- (4) Click the [Apply] button if you want to change other sections or properties or the [OK] button to end property settings. If you haven't clicked [Apply], you can use the [Revert Settings] button to discard the changes made and to return to the state in which this page was opened.

### [Edit Section] dialog box

#### Standard section



#### User section



Use the upper part of the dialog box to set the location information on a section in an execution file.

#### [Section name]

Set a user section name (output section).

**Note:** Keep in mind the following conditions when entering a section name:

- A section name must begin with ".".
- Only single-byte alphanumeric characters and symbols, the "\_" character, and "." are valid for section names.

#### [Virtual map address]

Set a location (start address) in which the section should be located when executed. To locate the section following another section, select the section immediately preceding name from the pull-down list. To locate the section at the beginning of a device or apart from the preceding section, enter the address of that location (in hexadecimal notation).

**Note:** When entering an address, use only 0–9 and A–F. Make sure that the address entered is six characters or less. Otherwise, your entry will be interpreted as a section name, not an address.

#### [LMA ≠ VMA]

Select this check box when the execution address (VMA) and stored address (LMA) of the section are different as for variables with initial values (e.g., .data section) and programs executed in RAM. Leave unselected if the section is to be executed directly in its stored location, as in ROM.

**[Load map address]**

If you selected [LMA ≠ VMA], set the address at which you want to store the section. If you want to locate the section following another section, select the section immediately preceding from the pull-down list. To locate the section at the beginning of a device or apart from the preceding section, enter the address of that location (in hexadecimal notation).

**Note:** When entering an address, use only 0–9 and A–F. Make sure that the address entered is six characters or less. Otherwise, your entry will be interpreted as a section name, not an address.

Use the lower part of the dialog box to set the location information on objects in the section.

The contents shown below may be changed only in user sections. The standard sections are predefined so that all object files for the project and the libraries already set in build options will be located. When a user section is defined, the settings of the standard section with the same attribute will be automatically updated to avoid overlaps.

**[File Attributes]**

Select an attribute from the pull-down list. Selectable attributes are the same as those of the standard sections.

The standard sections do not allow changing of the attribute.

**Unselected file list (left)**

Lists the object files of the project and the libraries already set in build options not located in this user section. Even before a build process, a list of object files is created from the source file names in the project. Select objects to be located in this section from the list.

In a standard section, this list may be blank or the objects that have been located in user sections with the same attribute are listed in unselectable status.

**Selected file list (right)**

Lists the object files and libraries located in this section.

In a standard section, all object files of the project and the libraries already set in build options locatable in this section (except those are located in user sections with the same attribute) are listed here. When linked, sections with the same attribute as that of an output section are extracted from within these files and listed in the output section in given order, beginning with the top of the list. The object files are listed in alphabetical order, after which the libraries are listed in the order in which the build options are set.

The list in standard sections is automatically updated according to the user section settings. It cannot be edited manually in contrast to user sections.

In user sections, the file configuration can be changed with the [<], [>], [Up], and [Down] buttons.

**[>]**

Selects a file to be located in this section from the unselected file list. This button is ineffective for standard sections.

**[<]**

Moves a file not to be located in this section from the selected file list to the unselected file list. This button is ineffective for standard sections.

**[Up]**

Changes the location of objects within the section. When you select an object in the selected file list and click this button, it swaps positions with the object immediately above it. This button is ineffective for standard sections.

**[Down]**

Changes the location of objects within the section. When you select an object in the selected file list and click this button, it swaps positions with the object immediately below it. This button is ineffective for standard sections.

**[Select All]**

Selects all entries in the respective lists. This button is ineffective for standard sections.

### Example of editing the section information (.vector section)

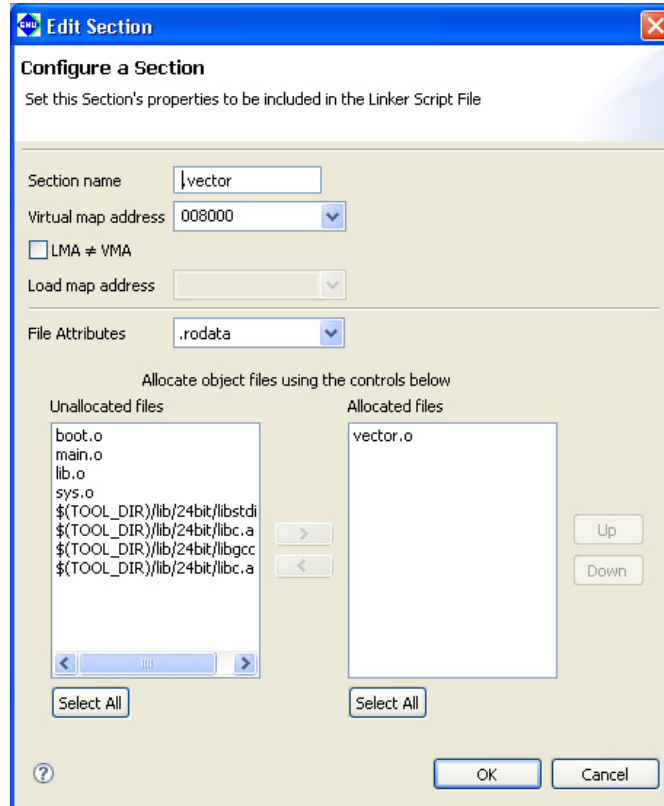
As an example of editing the section information, the .vector section information created with the default settings of the new project wizard is changed.

Section attribute: .rodata → .text

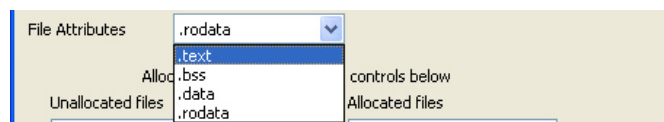
Object: vector.o → boot.o

- (1) In the [GNU17 Linker Script Settings] page for project properties, select the .vector section and click the [Edit] button.

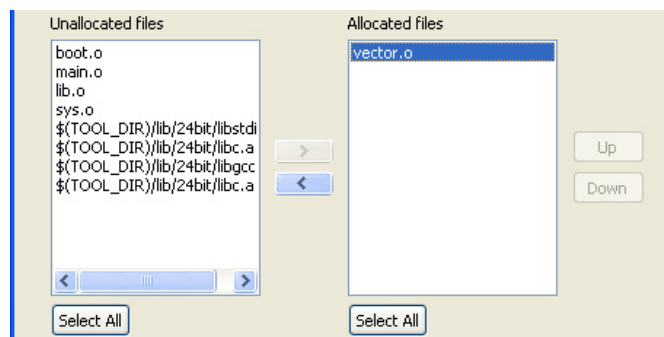
This displays the [Edit Section] dialog box.



- (2) Select .text from the [File Attributes] pull-down list.

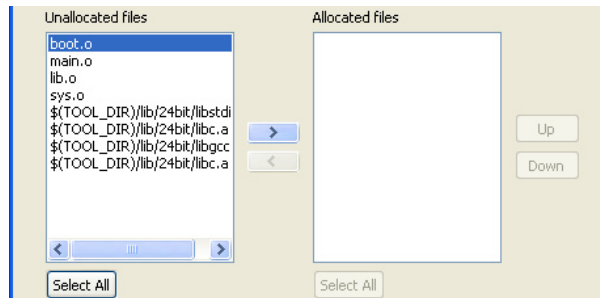


- (3) Select vector.o from the selected file list and click the [<] button.

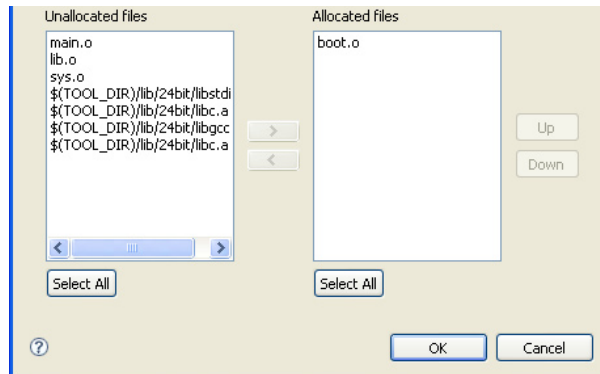


Since no source files are available for vector.o, it is removed without being moved to the unselected file list. This operation alone also removes vector.o from other section information.

- (4) Select `boot.o` from the unselected file list and click the [`>`] button.

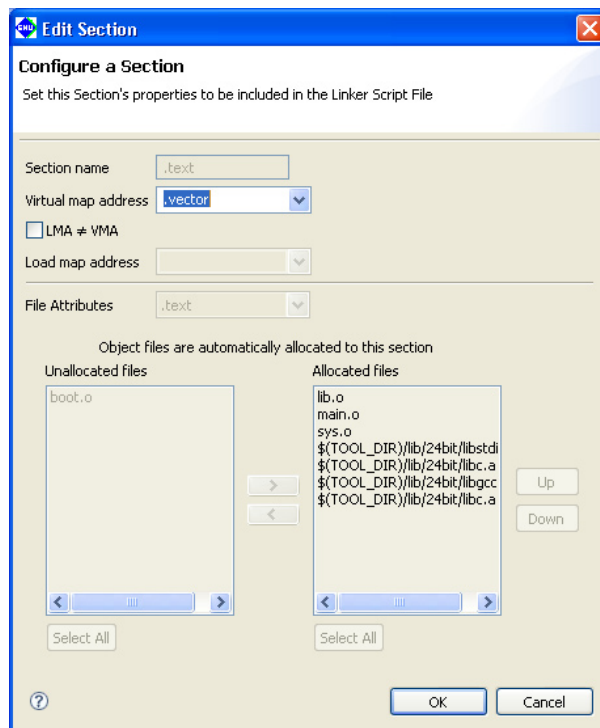


The selected `boot.o` is moved to the selected file list.



- (5) Click the [OK] button to end editing work.

Since the `.text` section of `boot.o` has been located in the `.vector` section, `boot.o` in the `.text` section information is moved to the unselected file list. (You cannot locate one file in multiple sections with the same attribute.)



- (6) Simply closing the [Edit Section] dialog box with the [OK] button will not automatically reflect the edits made here in the linker script. You must click the [Apply] or the [OK] button in the [Properties] dialog box to confirm your edits.

## Automatic updating of object files (for standard sections)

The standard section enables an automatic updating feature for the object files to be located in the sections. Since the linker script will automatically update itself whenever source files are added/removed, there is no need to manually reconfigure these sections.

The following describes the handling of object files in automatic updating.

### If no other sections have the same attribute

The object files for the project and the libraries already set in build options are all selected and located in that section.

### If there are multiple sections with the same attribute

The object files and libraries not located in other user sections with the same attribute are selected and located in that standard section. Files already located in user sections with the same attribute are not handled in automatic updating.

In user sections, the files included in other sections with the same attribute can also be added by selecting from the unselected file list. When a file is relocated from one section to another, it is removed from the section in which it was located up to that point.

The following limitations apply to standard sections in which automatic updating is enabled:

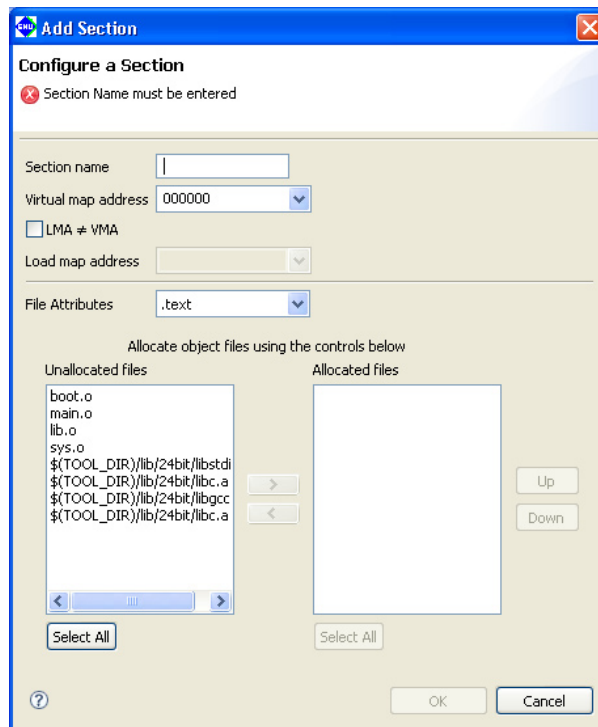
1. The section attribute ([File Attributes]) cannot be changed.
2. The file list can only be referenced; it cannot be manipulated.
3. Object files are added in ascending alphabetical order.

If these changes need to be made, create a user section.

## Adding a section

Follow the procedure described below to add a new section.

- (1) Click the [Add] button. This displays the [Add Section] dialog box.



- (2) Make the necessary settings based on the above discussion. Click [OK].

- (3) Click the [Apply] button to change other sections or properties or the [OK] button to end property settings.

If you haven't clicked [Apply], you can use the [Revert Settings] button to discard the changes made and restore the status in which this page was opened.

The section is inserted into the list according to the VMA you set.

To ensure that the new section location or objects will not overlap with another section, review the other section information, making alterations, if necessary.

## Removing a section

Do the following to remove unnecessary sections:

However, only user defined sections may be removed and standard sections cannot be removed.

- (1) In the [GNU17 Linker Script Settings] page, click to select the section you want to remove from the section list.
- (2) Click the [Delete] button.
- (3) A dialog box for confirmation will be displayed. Click [OK] to delete or [Cancel] to cancel.
- (4) Click the [Apply] button if you want to change other sections or properties or the [OK] button to end property settings.

If you haven't clicked [Apply], you can use the [Revert Settings] button to discard the changes made and restore the status in which this page was opened.

### \* About changes in referenced sections pursuant to deletion

```
Section1 ([Virtual map address] = 000000)
Section2 ([Virtual map address] = Section1)
Section3 ([Virtual map address] = Section2)
Section4 ([Virtual map address] = Section3)
```

Removing Section2 in a section configuration like the one shown will automatically change Section3 so that Section1 is referenced instead.

```
Section1 ([Virtual map address] = 000000)
Section3 ([Virtual map address] = Section1)
Section4 ([Virtual map address] = Section3)
```

If Section1 is removed, the location address of Section2 changes to (0x)000000. If this results in problems, reedit the section information.

```
Section2 ([Virtual map address] = 000000)
Section3 ([Virtual map address] = Section2)
Section4 ([Virtual map address] = Section3)
```

## Precautions

- No more than 255 characters may be entered for each of [Section name], [Virtual map address], and [Load map address] in the [Edit Section] dialog box.
- A section name must begin with ".". Only single-byte alphanumeric characters, "\_", and "." are valid for section names.
- When entering an address in [Virtual map address] or [Load map address], use only 0–9 and A–F. Make sure that the address entered is six characters or less. Otherwise, your entry will be interpreted as a section name, not an address.
- Although the **IDE** checks backward and cyclic references from one section to another, this check is not necessarily complete, and an error may result at linking.
- The object file (`vector.o`) to be mapped to the `.vector` section, which was specified in the [New GNU17 Project] wizard, must be located in the project directory. The `.vector` section settings in the linker script must be edited when the object file is located in another directory.



## Example linker script settings

Shown below are example linker script settings for several section configurations using sample screens. For more information on making these settings, refer to the discussion on the preceding pages.

Example 1: Minimum section configuration

Example 2: Changing the basic layout

Example 3: Sharing the RAM area with multiple variables

Example 4: Executing a program in RAM

It is assumed that the vector table in the assembler source (`boot.o`) is written in the `.rodata` section and that the table is set in the new project wizard to be located in the `.vector` section.

### Example 1: Minimum section configuration

Described here is a system with the simplest possible configuration using RAM and ROM.

The program and data will be located in the ROM beginning with address `0x8000` as shown in Figure 5.7.6.2. The program is assumed to run directly from its stored ROM address (LMA), and static data is also assumed to be read out for use directly from the ROM. The variable without initial values will be located from address `0x0` in the RAM. Subsequent areas are used for variables with initial values. The initial values of variables are stored in the ROM and copied from the ROM into the RAM by the application program.

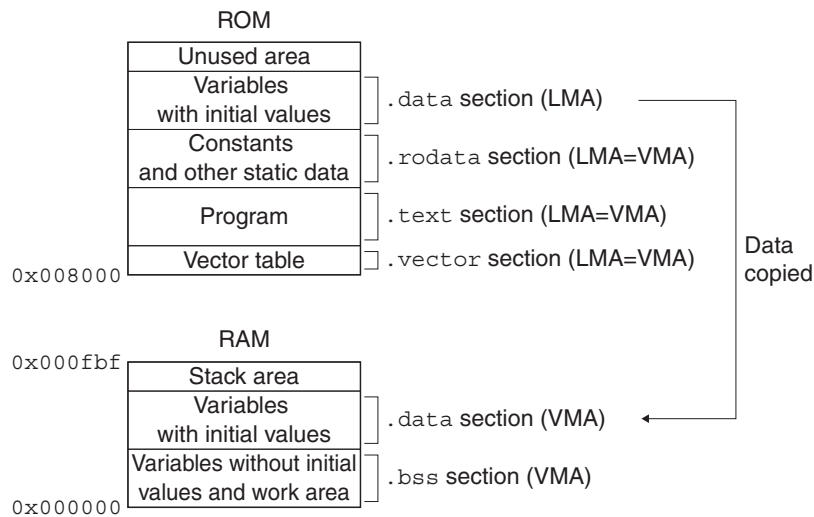


Figure 5.7.6.2 Example of a memory configuration 1

Using default linker script files is the simplest method for using memory in this way.

This section location can be realized without adding to or correcting settings in the [GNU17 Linker Script Settings] dialog box.

### Example of a source file configuration

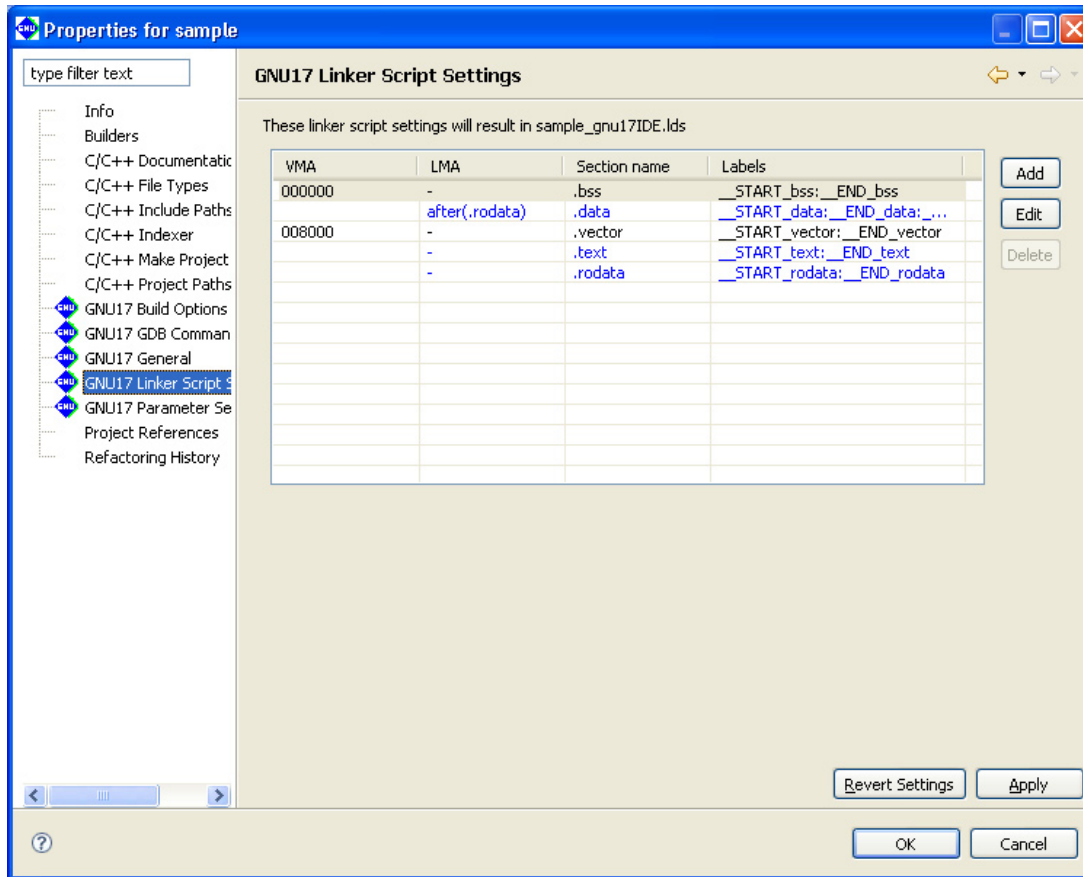
`boot.s` (vector table and stack initialization, etc.)

`main.c` (main and other functions)

The program is assumed to be comprised of these two sources.

Except when the location addresses of the respective sections are specified individually, the sections when linked are located in alphabetical order of file names. In the example here, the sections are located in order of `boot.s` and `main.c`. The vector table at the beginning of `boot.s` (i.e., the `.rodata` section) is placed at the beginning of the ROM (`0x8000` and beyond).

## Section configuration (contents set in the [GNU17 Linker Script Settings] dialog box)



These are the contents set by default.

The VMAs (execution addresses) are set so that the `.bss` section is located from address 0x0 (RAM) and the `.vector` section (`.rodata` section of `boot.o`) is located from address 0x8000 (start of ROM). Other sections are located at addresses following these two sections. Since sections other than `.data` are used from their stored addresses, no LMAs (load addresses) are set. Since `.data` is copied from ROM to RAM before use, a LMA is set following the `.rodata` section. The memory map configuration in Figure 5.7.6.2 can be realized directly, without modifying this section configuration.

### Linker script and section location

The linker script is generated as shown below.

```
/* Linker Script file generated by Gnu17 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);

SECTIONS
{
    /* location counter */
    . = 0x0;

    /* section information */
    .bss 0x000000 :
    {
        __START_bss = . ;
        boot.o(.bss)
        main.o(.bss)
        C:/EPSON/gnu17/lib/24bit/libstdio.a(.bss)
        C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
        C:/EPSON/gnu17/lib/24bit/libgcc.a(.bss)
        C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
        __END_bss = . ;
    }
}
```

```

.data __END_bss : AT( __END_rodata )
{
  __START_data = . ;
  boot.o(.data)
  main.o(.data)
  C:/EPSON/gnu17/lib/24bit/libstdio.a(.data)
  C:/EPSON/gnu17/lib/24bit/libc.a(.data)
  C:/EPSON/gnu17/lib/24bit/libgcc.a(.data)
  C:/EPSON/gnu17/lib/24bit/libc.a(.data)
  __END_data = . ;
}

.vector 0x008000 :
{
  __START_vector = . ;
  boot.o(.rodata)
  __END_vector = . ;
}

.text __END_vector :
{
  __START_text = . ;
  boot.o(.text)
  main.o(.text)
  C:/EPSON/gnu17/lib/24bit/libstdio.a(.text)
  C:/EPSON/gnu17/lib/24bit/libc.a(.text)
  C:/EPSON/gnu17/lib/24bit/libgcc.a(.text)
  C:/EPSON/gnu17/lib/24bit/libc.a(.text)
  __END_text = . ;
}

.rodata __END_text :
{
  __START_rodata = . ;
  boot.o(.rodata)
  main.o(.rodata)
  C:/EPSON/gnu17/lib/24bit/libstdio.a(.rodata)
  C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
  C:/EPSON/gnu17/lib/24bit/libgcc.a(.rodata)
  C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
  __END_rodata = . ;
}

/* load address symbols */
__START_data_lma = LOADADDR( .data );
__END_data_lma = __START_data_lma + SIZEOF( .data );
}

```

The section location including a file configuration is shown below.

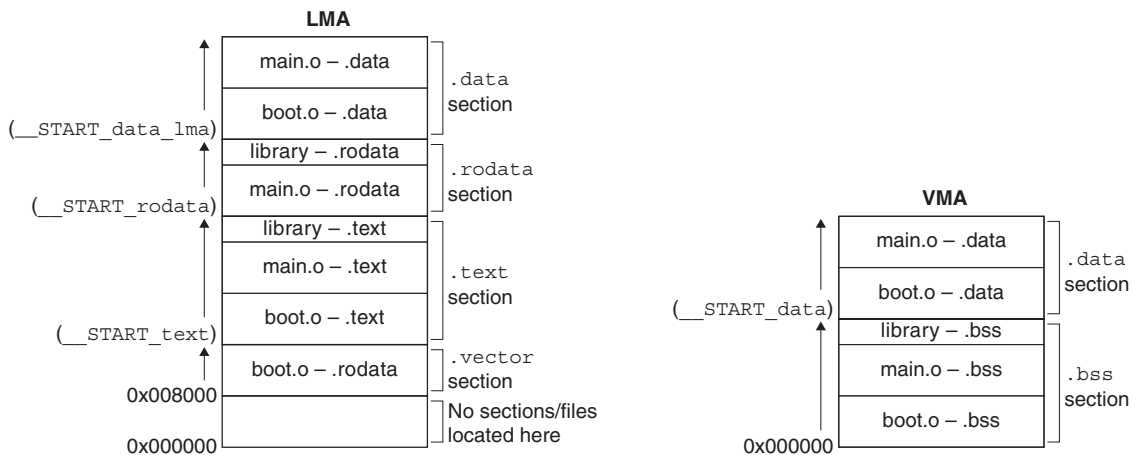


Figure 5.7.6.3 Example of a section location 1

**Example 2: Changing the basic layout**

Here, a ROM for storing constants is added to the system in Example 1. Shown below is an example of how to locate the .rodata section.

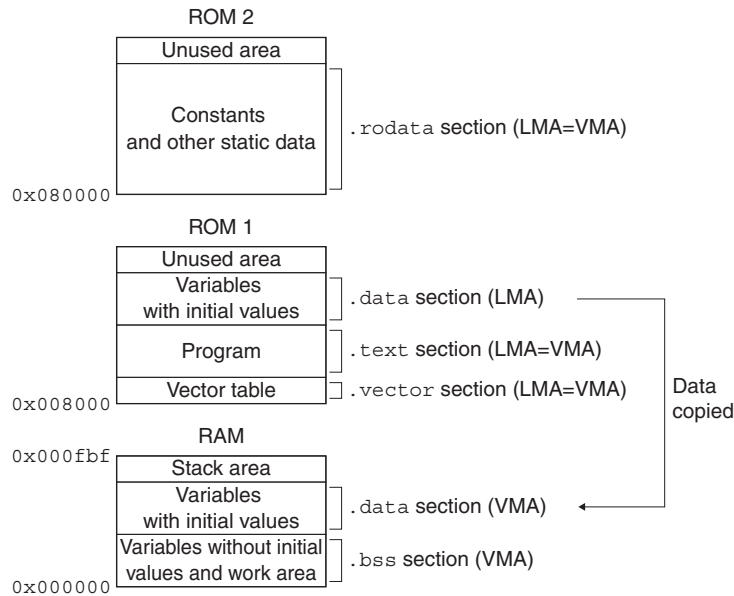


Figure 5.7.6.4 Example of a memory configuration 2

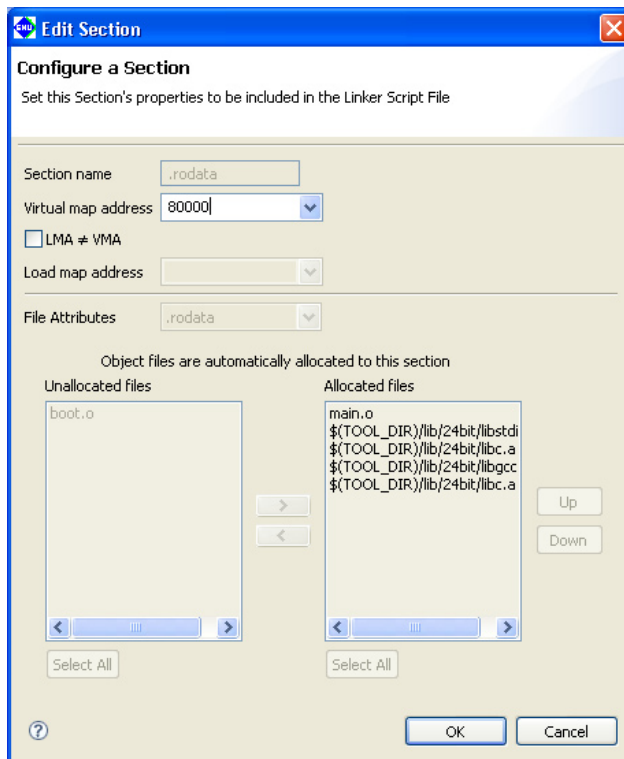
**Example of a source file configuration**

boot.s (vector table and stack initialization, etc.)  
 main.c (main and other functions)

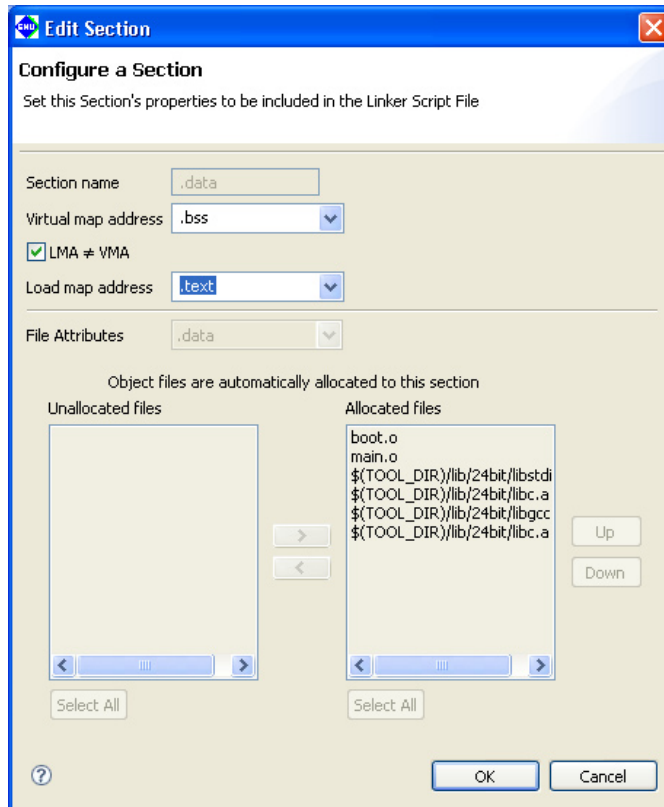
**Editing of sections (contents set in the [Edit Section] dialog box)**

Correct the .rodata and .data section information as shown below. Use all other sections with default settings directly and unaltered.

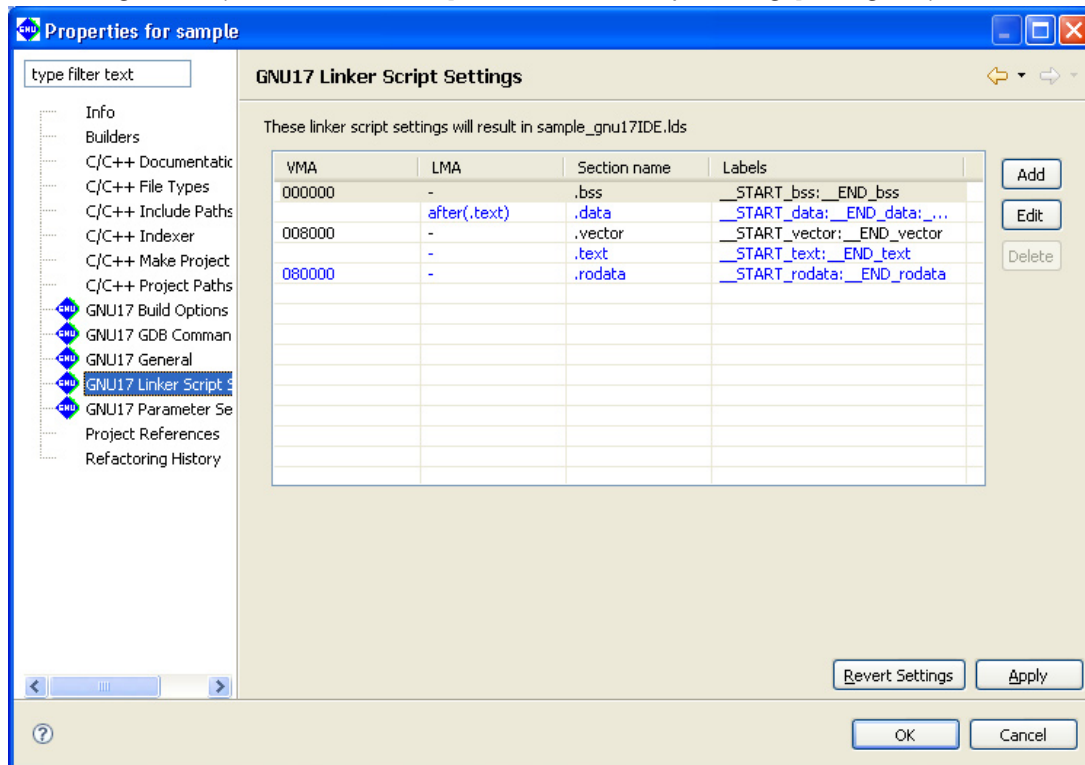
1. Correcting the .rodata section  
 Correct [Virtual map address] to 0x080000.



- Correcting the `.data` section  
Correct [Load map address] to `.text`.



Section configuration (contents set in the [GNU17 Linker Script Settings] dialog box)



## Linker script and section location

The linker script is generated as shown below.

```

/* Linker Script file generated by Gnu17 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);

SECTIONS
{
/* location counter */
. = 0x0;

/* section information */
.bss 0x000000 :
{
__START_bss = . ;
boot.o(.bss)
main.o(.bss)
C:/EPSON/gnu17/lib/24bit/libstdio.a(.bss)
C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
C:/EPSON/gnu17/lib/24bit/libgcc.a(.bss)
C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
__END_bss = . ;
}

.data __END_bss : AT( __END_text )
{
__START_data = . ;
boot.o(.data)
main.o(.data)
C:/EPSON/gnu17/lib/24bit/libstdio.a(.data)
C:/EPSON/gnu17/lib/24bit/libc.a(.data)
C:/EPSON/gnu17/lib/24bit/libgcc.a(.data)
C:/EPSON/gnu17/lib/24bit/libc.a(.data)
__END_data = . ;
}

.vector 0x008000 :
{
__START_vector = . ;
boot.o(.rodata)
__END_vector = . ;
}

.text __END_vector :
{
__START_text = . ;
boot.o(.text)
main.o(.text)
C:/EPSON/gnu17/lib/24bit/libstdio.a(.text)
C:/EPSON/gnu17/lib/24bit/libc.a(.text)
C:/EPSON/gnu17/lib/24bit/libgcc.a(.text)
C:/EPSON/gnu17/lib/24bit/libc.a(.text)
__END_text = . ;
}

.rodata 0x080000 :
{
__START_rodata = . ;
main.o(.rodata)
C:/EPSON/gnu17/lib/24bit/libstdio.a(.rodata)
C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
C:/EPSON/gnu17/lib/24bit/libgcc.a(.rodata)
C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
__END_rodata = . ;
}

```

```

/* load address symbols */
__START_data_lma = LOADADDR( .data );
__END_data_lma = __START_data_lma + SIZEOF( .data );
}

```

Shown below are section locations and file configurations.

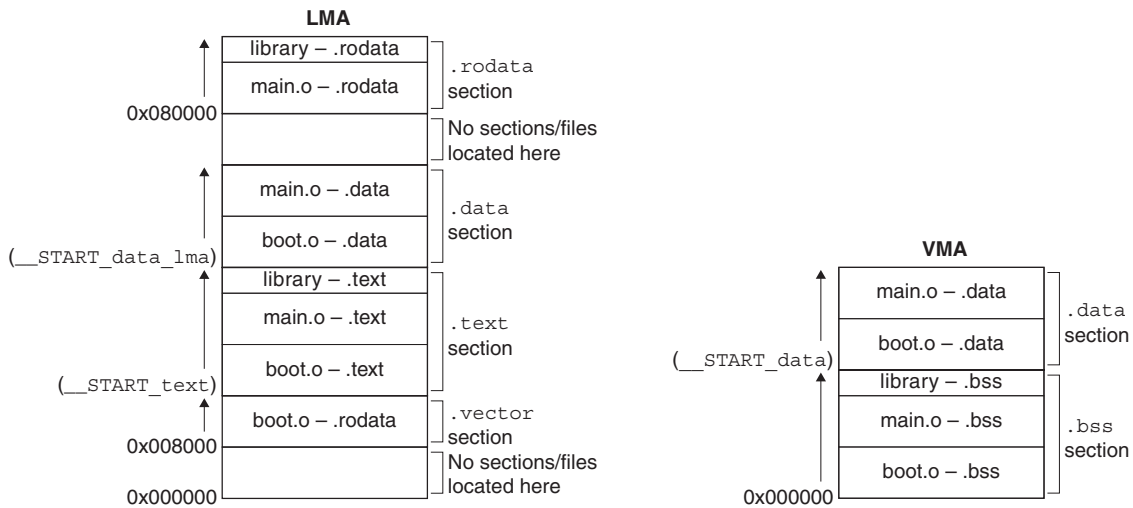


Figure 5.7.6.5 Example of section location 2

### Example 3. Sharing the RAM area with multiple variables

Shown below is an example of how to assign the same RAM area to multiple variables in a system with the memory configuration shown in Example 1.

Multiple sections are allocated to the same address as in Figure 5.7.6.6, where the same data is shared by multiple variables and the data for one section is exchanged for another when used. This permits efficient use of memory. However, only sections having the `.bss` attribute can share data areas. The area set aside for variables with initial values (`.data` section) cannot be shared.

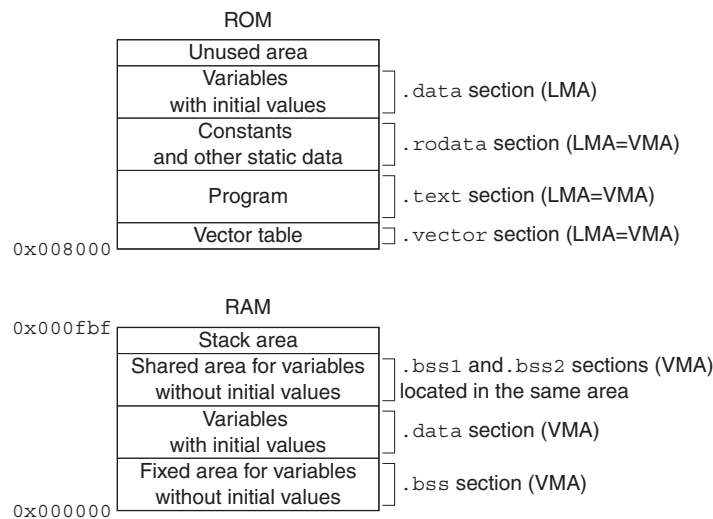


Figure 5.7.6.6 Example for shared data area

Example of a source file configuration

```

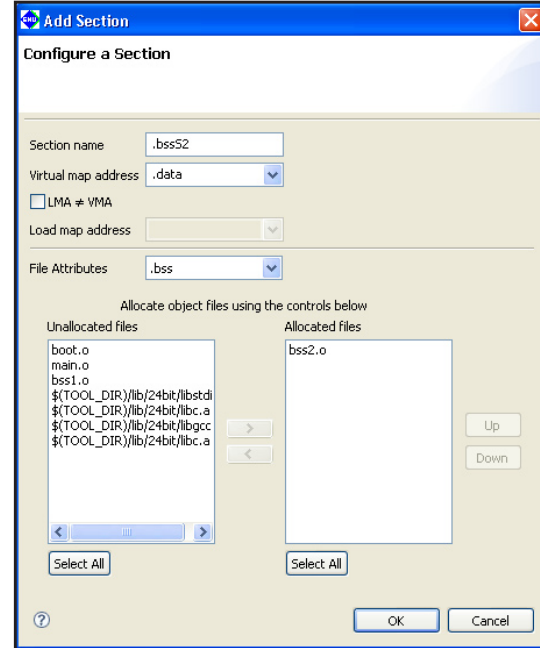
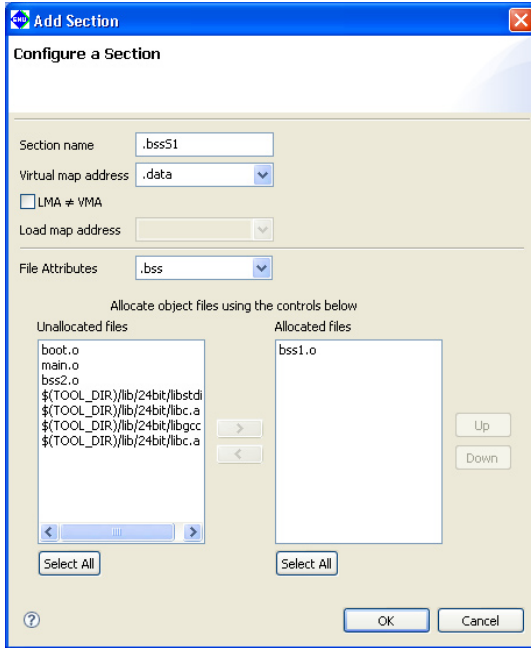
boot.s (vector table and stack initialization, etc.)
main.c (main and other functions)
bss1.c (global variable definition file 1)
bss2.c (global variable definition file 2)

```

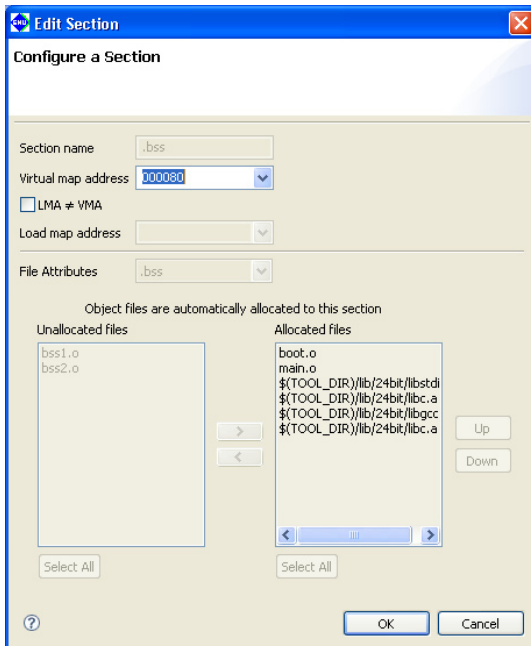
`bss1.c` and `bss2.c` are assumed to consist only of a definition of global variables without initial values that share an area. If these files contain functions, variables with initial values, or constants, the files in the example here will be located in the `.text`, `.data`, or `.rodata` sections.

Editing sections (content set in the [Add/Edit Section] dialog box)

Create new sections `.bssS1` and `.bssS2` that share an area of RAM. Both sections assume the `.bss` attribute and are located immediately after the `.data` section (VMA). As files for the respective sections, select only `bss1.o` for the `.bssS1` section and only `bss2.o` for the `.bssS2` section.

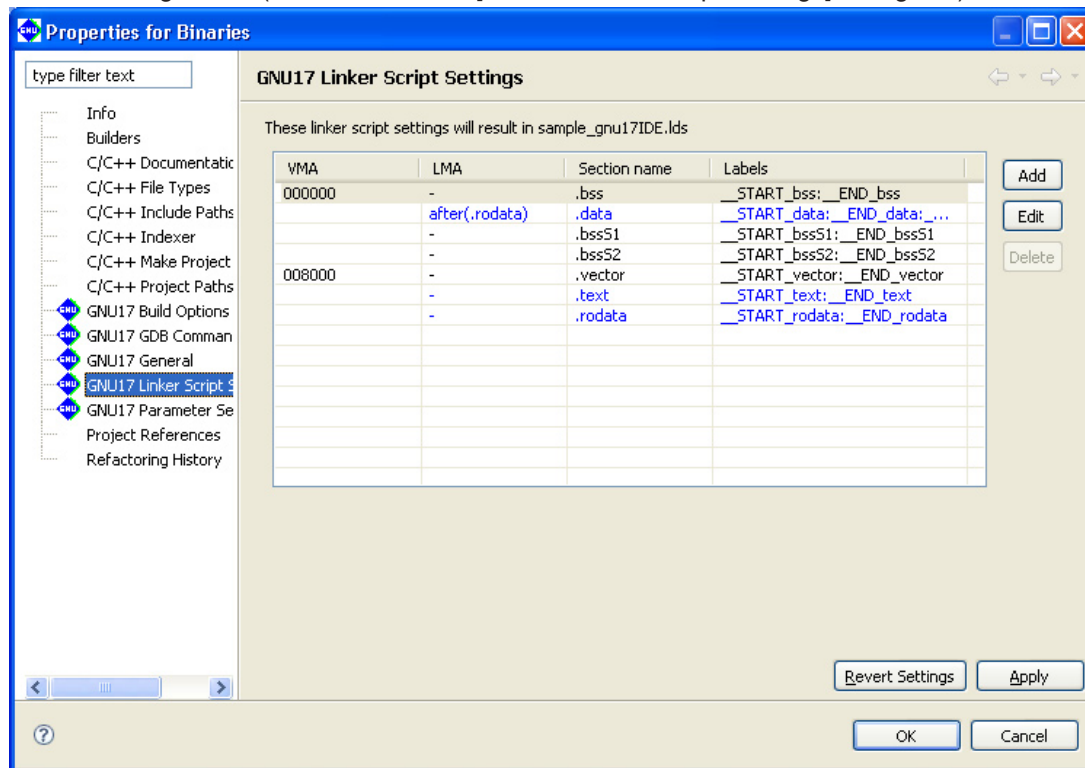


Note that automatic updating is enabled for `.bss` section information and that `bss1.o` and `bss2.o` are not included in the list of files to be located. This is because they have been specified for the respective sections above.





Section configuration (content set in the [GNU17 Linker Script Settings] dialog box)



### Linker script and section location

The linker script is generated as shown below.

```

/* Linker Script file generated by Gnu17 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);

SECTIONS
{
    /* location counter */
    . = 0x0;

    /* section information */
    .bss 0x000000 :
    {
        __START_bss = . ;
        boot.o(.bss)
        main.o(.bss)
        C:/EPSON/gnu17/lib/24bit/libstdio.a(.bss)
        C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
        C:/EPSON/gnu17/lib/24bit/libgcc.a(.bss)
        C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
        __END_bss = . ;
    }

    .data __END_bss : AT( __END_rodata )
    {
        __START_data = . ;
        boot.o(.data)
        bss1.o(.data)
        bss2.o(.data)
        main.o(.data)
        C:/EPSON/gnu17/lib/24bit/libstdio.a(.data)
        C:/EPSON/gnu17/lib/24bit/libc.a(.data)
        C:/EPSON/gnu17/lib/24bit/libgcc.a(.data)
        C:/EPSON/gnu17/lib/24bit/libc.a(.data)
        __END_data = . ;
    }
}

```

```

.bssS1 __END_data :
{
  __START_bssS1 = . ;
  bss1.o(.bss)
  __END_bssS1 = . ;
}

.bssS2 __END_data :
{
  __START_bssS2 = . ;
  bss2.o(.bss)
  __END_bssS2 = . ;
}

.vector 0x008000 :
{
  __START_vector = . ;
  boot.o(.rodata)
  __END_vector = . ;
}

.text __END_vector :
{
  __START_text = . ;
  boot.o(.text)
  bss1.o(.text)
  bss2.o(.text)
  main.o(.text)
  C:/EPSON/gnu17/lib/24bit/libstdio.a(.text)
  C:/EPSON/gnu17/lib/24bit/libc.a(.text)
  C:/EPSON/gnu17/lib/24bit/libgcc.a(.text)
  C:/EPSON/gnu17/lib/24bit/libc.a(.text)
  __END_text = . ;
}

.rodata __END_text :
{
  __START_rodata = . ;
  bss1.o(.rodata)
  bss2.o(.rodata)
  main.o(.rodata)
  C:/EPSON/gnu17/lib/24bit/libstdio.a(.rodata)
  C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
  C:/EPSON/gnu17/lib/24bit/libgcc.a(.rodata)
  C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
  __END_rodata = . ;
}

/* load address symbols */
__START_data_lma = LOADADDR( .data );
__END_data_lma = __START_data_lma + SIZEOF( .data );
}

```

Shown below are section locations and file configurations.

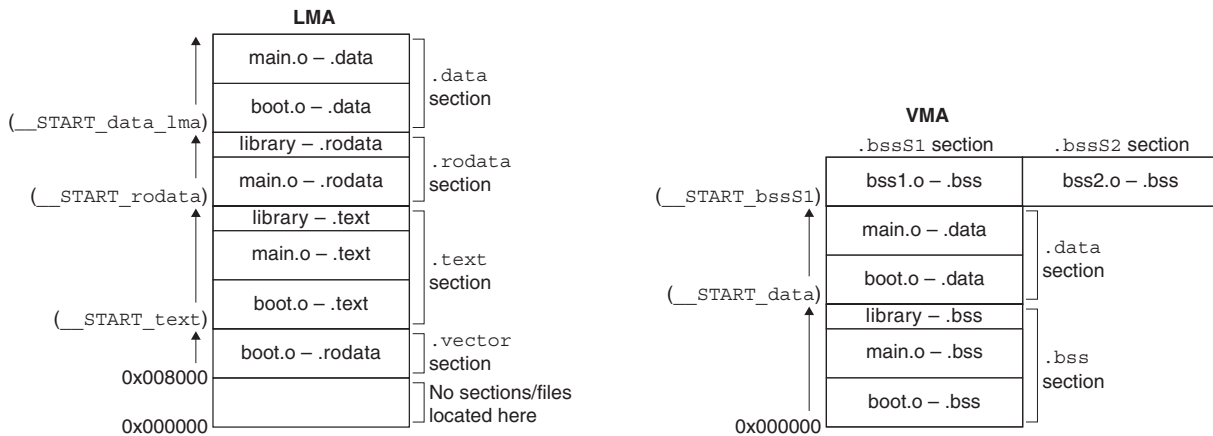


Figure 5.7.6.7 Example of a section location 3

The `.bssS1` and `.bssS2` sections are located in parallel. The application determines which section is managed and what data is used.

#### Example 4: Executing a program in RAM

A routine that requires high-speed processing can be executed in RAM that can be accessed without wait states to meet specific requirements. In the examples seen so far, only the VMA of the `.text` section is specified, and the program is executed in ROM. However, the program can be executed in RAM by first specifying the VMA and LMA, as for the `.data` section, then copying the program to RAM before execution. Additionally, multiple sections may be allocated to the same area, as in Example 3, and the program may be executed by exchanging sections as necessary.

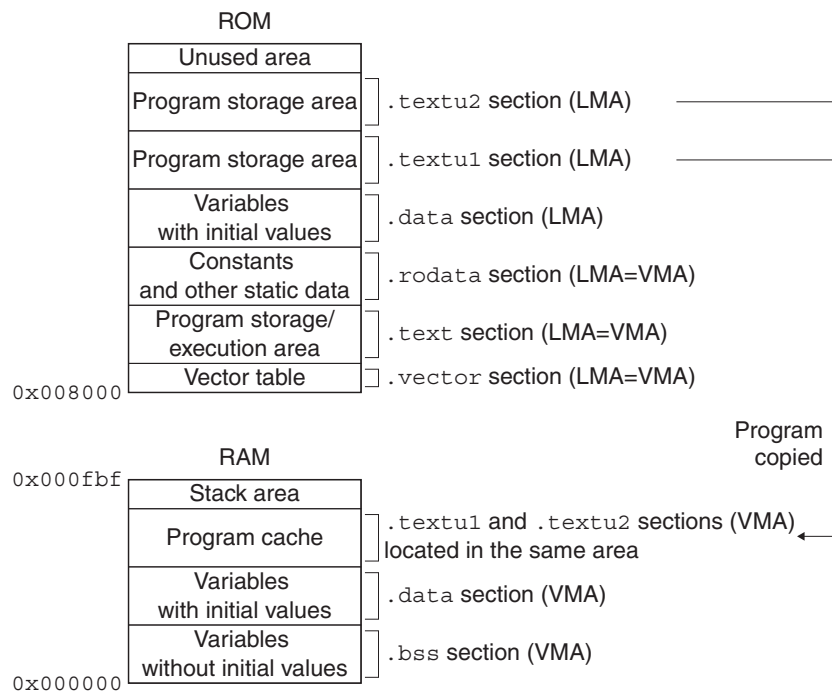


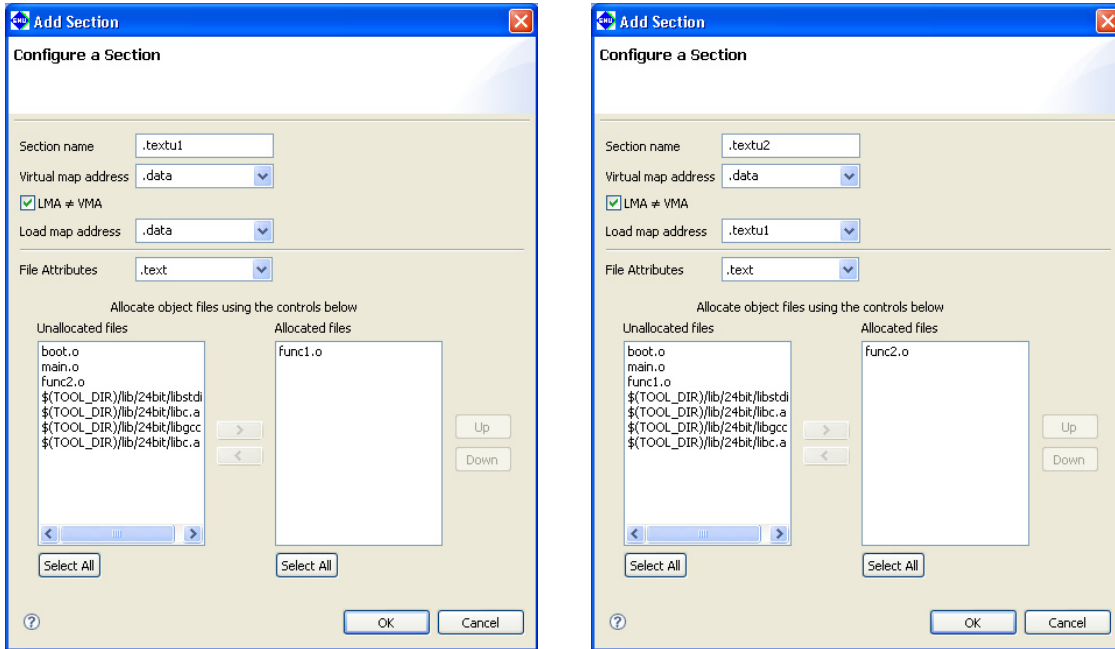
Figure 5.7.6.8 Allocating a storage area for the program in RAM

#### Example of a source file configuration

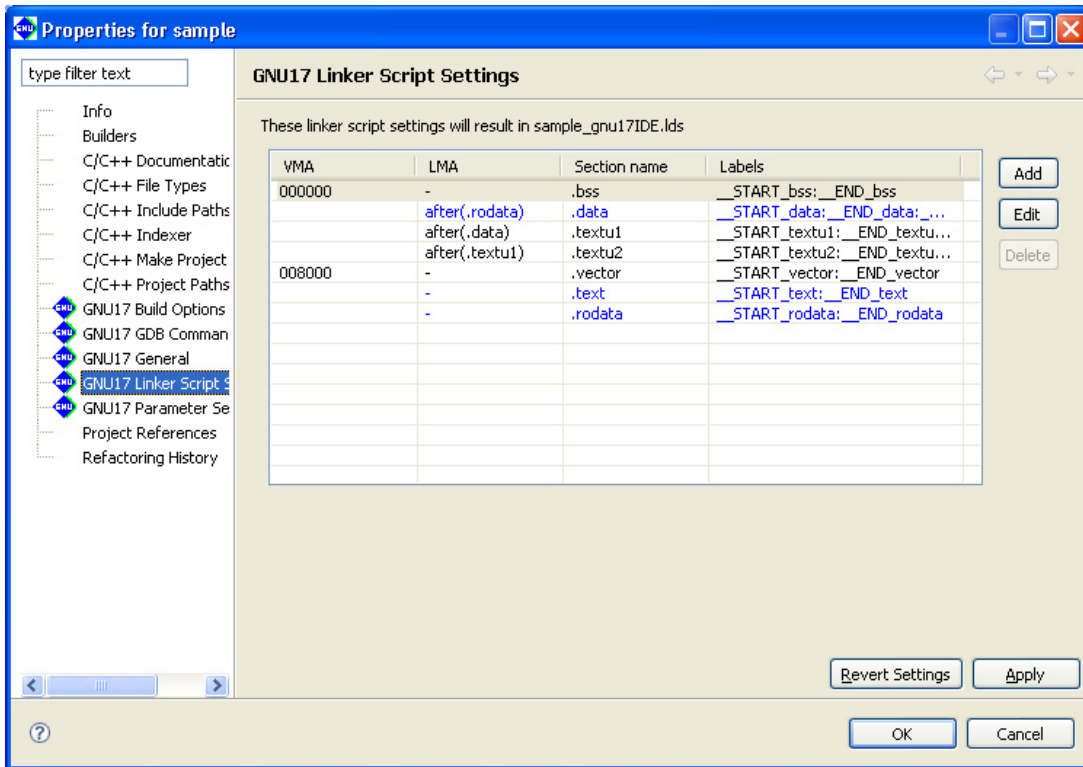
```
boot.s (vector table and stack initialization, etc.)
main.c (main and other functions)
func1.c (program 1 to be executed in RAM)
func2.c (program 2 to be executed in RAM)
```

Editing sections (content set in the [Add Section] dialog box)

Create new sections `.textu1` and `.textu2`, as shown below.



Section configuration (content set in the [GNU17 Linker Script Settings] dialog box)



## Linker script and section location

The linker script is generated, as shown below.

```

/* Linker Script file generated by Gnu17 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);

SECTIONS
{
  /* location counter */
  . = 0x0;

  /* section information */
  .bss 0x000000 :
  {
    __START_bss = . ;
    boot.o(.bss)
    func1.o(.bss)
    func2.o(.bss)
    main.o(.bss)
    C:/EPSON/gnu17/lib/24bit/libstdio.a(.bss)
    C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
    C:/EPSON/gnu17/lib/24bit/libgcc.a(.bss)
    C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
    __END_bss = . ;
  }

  .data __END_bss : AT( __END_rodata )
  {
    __START_data = . ;
    boot.o(.data)
    func1.o(.data)
    func2.o(.data)
    main.o(.data)
    C:/EPSON/gnu17/lib/24bit/libstdio.a(.data)
    C:/EPSON/gnu17/lib/24bit/libc.a(.data)
    C:/EPSON/gnu17/lib/24bit/libgcc.a(.data)
    C:/EPSON/gnu17/lib/24bit/libc.a(.data)
    __END_data = . ;
  }

  .text1 __END_data : AT( __START_data_lma + SIZEOF( .data ) )
  {
    __START_text1 = . ;
    func1.o(.text)
    __END_text1 = . ;
  }

  .text2 __END_data : AT( __START_text1_lma + SIZEOF( .text1 ) )
  {
    __START_text2 = . ;
    func2.o(.text)
    __END_text2 = . ;
  }

  .vector 0x008000 :
  {
    __START_vector = . ;
    boot.o(.rodata)
    __END_vector = . ;
  }

  .text __END_vector :
  {
    __START_text = . ;
    boot.o(.text)
    main.o(.text)
  }
}

```

```

C:/EPSON/gnu17/lib/24bit/libstdio.a(.text)
C:/EPSON/gnu17/lib/24bit/libc.a(.text)
C:/EPSON/gnu17/lib/24bit/libgcc.a(.text)
C:/EPSON/gnu17/lib/24bit/libc.a(.text)
__END_text = . ;
}

.rodata __END_text :
{
    __START_rodara = . ;
    func1.o(.rodara)
    func2.o(.rodara)
    main.o(.rodara)
    C:/EPSON/gnu17/lib/24bit/libstdio.a(.rodara)
    C:/EPSON/gnu17/lib/24bit/libc.a(.rodara)
    C:/EPSON/gnu17/lib/24bit/libgcc.a(.rodara)
    C:/EPSON/gnu17/lib/24bit/libc.a(.rodara)
    __END_rodara = . ;
}

/* load address symbols */
__START_data_lma = LOADADDR( .data );
__END_data_lma = __START_data_lma + SIZEOF( .data );
__START_textu1_lma = LOADADDR( .textu1 );
__END_textu1_lma = __START_textu1_lma + SIZEOF( .textu1 );
__START_textu2_lma = LOADADDR( .textu2 );
__END_textu2_lma = __START_textu2_lma + SIZEOF( .textu2 );
}

```

Shown below are the section locations and file configurations.

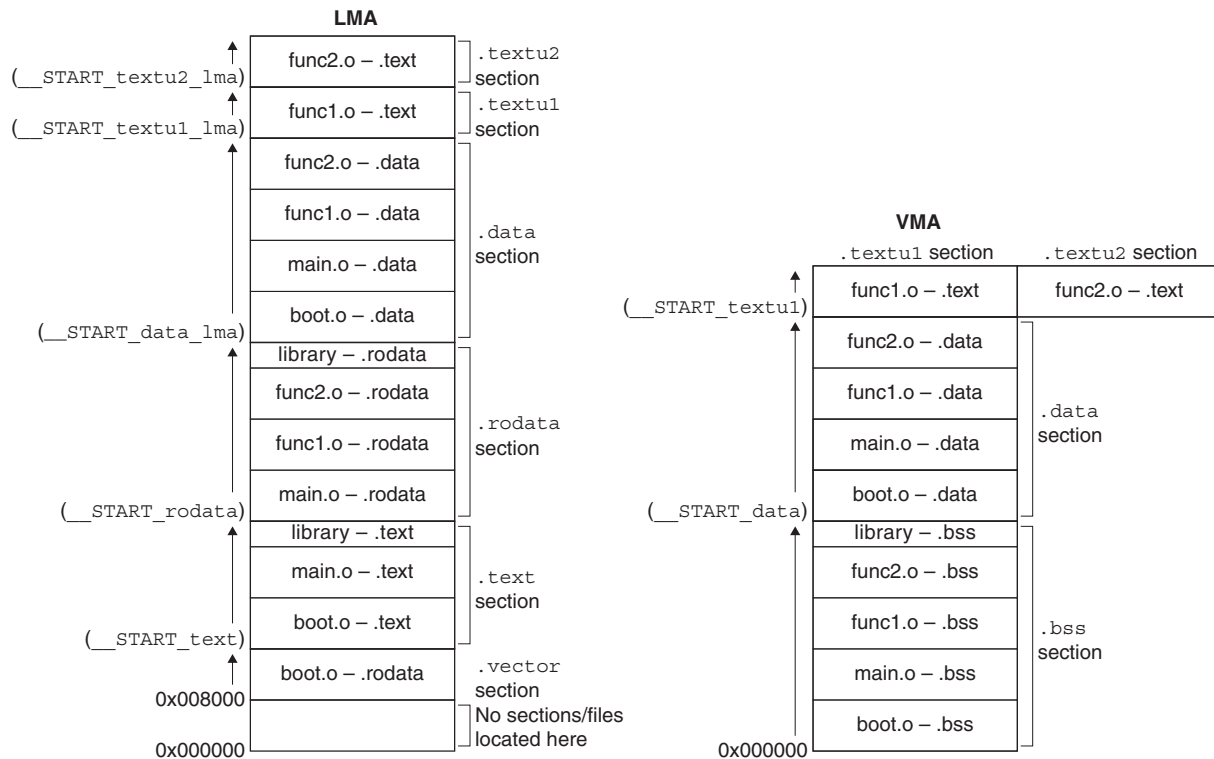


Figure 5.7.6.9 Example of s section location 4

The routine for transferring the program from ROM to RAM should be created within main.c.

**Note:** Locating the .textu1 and .textu2 sections preceding a .rodara section with (for example) no LMA specified will result in positioning the .rodara section behind the end VMA address of the .textu2 section. Do not locate any VMA-only sections behind LMA ≠ VMA sections.

## 5.7.7 Executing a Build Process

After creating source files, setting build options, and editing a linker script file, you can execute a build process. Shown below is the procedure for executing a build process.

### Building all projects in the workspace

Do one of the following to build all projects present in the workspace:

- Select [Build All] from the [Project] menu.
- Click the [Build All] button in the window toolbar.

### Building a selected project

Do the following to build a project individually:

- (1) Select the project you want to build in the [C/C++ Projects] or [Navigator] view.
- (2) Do the following to execute a build process:
  - Select [Build Project] from the [Project] menu.
  - Select [Build Project] from the context menu in the [C/C++ Projects] or [Navigator] view.

You also can select a working set from [Build Working Set] on the [Project] menu and build only projects included in the selected working set.

### Build process

When you begin building a project, the **IDE** executes the processing described below.

1. Save any unsaved files in the editor.
2. Generate the following files according to the settings for project properties:
  - Makefile (*<project name>\_gnu17IDE.mak*) and dependency file (*<source name>.d*)
  - Linker script file (*<project name>\_gnu17IDE.lds*)
  - Parameter file (*<project name>\_gnu17IDE.par*)\*
  - Command file (*<project name>\_gnu17IDE.cmd*)\*
    - \* These files are needed for debugging and do not affect the build process. Normally, no command file is generated in a build process. A build process generates a command file that includes a minimum command set required for starting up the debugger only when *<project name>\_gnu17IDE.cmd* does not exist.
3. Execute **make.exe**. The following files will be generated:
  - Object file for each source (*<source name>.o*)
  - Executable format object file (*<project name>.elf*)
  - Link map file (*<project name>.map*)
  - Dump file (*<project name>.dump*)

While a build process is underway, the command line in [Console] view shows each tool being executed.

Any errors occurring during a build process can be reviewed in [Problems] view. From there, you can jump the corresponding spot in the editor in error. For more information on this feature, refer to "Jump to a line with an error" in Section 5.5.3.

Note that **make.exe** is designed to generate and link only the object files (\*.o) that have yet to be generated or that require updating by checking the dependency list of the source files and object files (\*.o) written in the makefile.

Therefore, in the first build process, **make.exe** compiles/assembles all sources to generate object files (\*.o) and to link the generated files. Thereafter, it compiles/assembles only the altered sources (including alteration of include files) and links the generated files.

The files listed in 2 above require caution. Even if you correct their contents directly in the editor before a build process, they will be overwritten when a build process starts. To ensure that these files are not regenerated during a build process, deselect the [GNU17 File Builder] currently selected check box in the [Builders] page of the [Properties] dialog box.

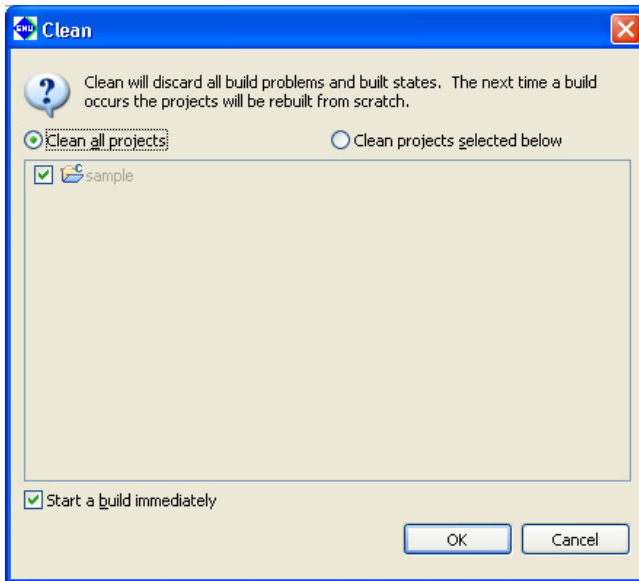
## 5.7.8 Clean and Rebuild

As described above, no object files (\*.o) are regenerated unless the source or include files have been altered. If all of the generated object files (\*.o) are erased, a build (or rebuild) from all sources can be re-executed. Therefore, the makefile generated by the **IDE** has a command for erasing all generated files written in it with the target name "clean".

The following describes the procedure for rebuilding a project using this command:

- (1) Select the project you want to rebuild in the [C/C++ Projects] or [Navigator] view.
- (2) Select [Clean...] from the [Project] menu.

This displays the [Clean] dialog box.



- (3) Select the [Clean projects selected below] radio button and select the project to execute "clean" (rebuild) from the list.

Select [Clean all projects] to rebuild all projects in the workspace.

Leave the [Start a build immediately] check box selected. This allows you to proceed to a build process directly, without doing anything, else after executing the clean command.

- (4) Click the [OK] button.

This deletes all generated object files in the selected project, then executes a build process.

If the [Start a build immediately] check box is unselected, the command will only perform the file deletions. You must execute the build process as described in the preceding section.

You also can also execute "clean" as described below:

- (1) Select the project you want to execute "clean" in the [C/C++ Projects] view.
- (2) Select [Clean Project] from the context menu in the [C/C++ Projects] view.

In this case, no dialog boxes are displayed, and the "clean" process only is executed.

Except when you intend to rebuild a project, you will need to execute a build process after altering certain source files or header files. You must perform a rebuild in the following cases:

- When certain header files have been deleted from the project  
The dependency file is regenerated by rebuilding the project.
- When the dependency file is deleted  
Do not delete the dependency file.
- When the included header file does not exist in the project  
You must rebuild if you've altered a header file not existing in the project file (i.e., external to the project).



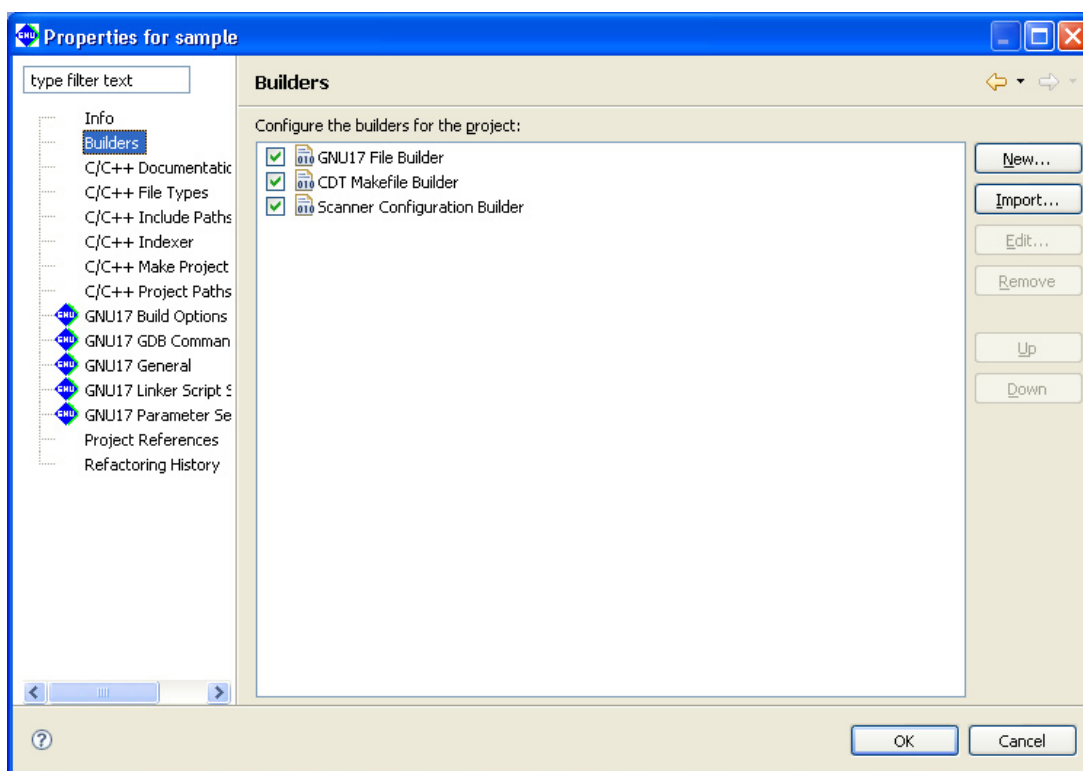
## 5.7.9 Using an Original Makefile

To perform a build, the original makefiles created by a user may be used instead of those automatically generated by the **IDE**. This procedure is described below.

- (1) In the [C/C++ Projects] or [Navigator] view, select the project you want to build using the makefile you created.
- (2) Create a new makefile in the **IDE** or import the makefile you already created. (Refer to Section 5.4.8, "Resource Manipulation in a Project.")  
If you've already performed a build in the **IDE**, you can use the makefile generated by the **IDE** after correcting it.
- (3) Select [Properties] from the [Project] menu or the context menu of the selected project.  
This displays the [Properties] dialog box.

If you created a makefile with the name "*<project name>\_gnu17IDE.mak*" or use a corrected version of the **IDE**-generated makefile, steps (4) and (5) described below are required. If you are using a makefile with a different name and use the **IDE**-generated files for the linker script file (.lds), parameter file (.par), and command file (.cmd), skip steps (4) and (5) and go to (6).

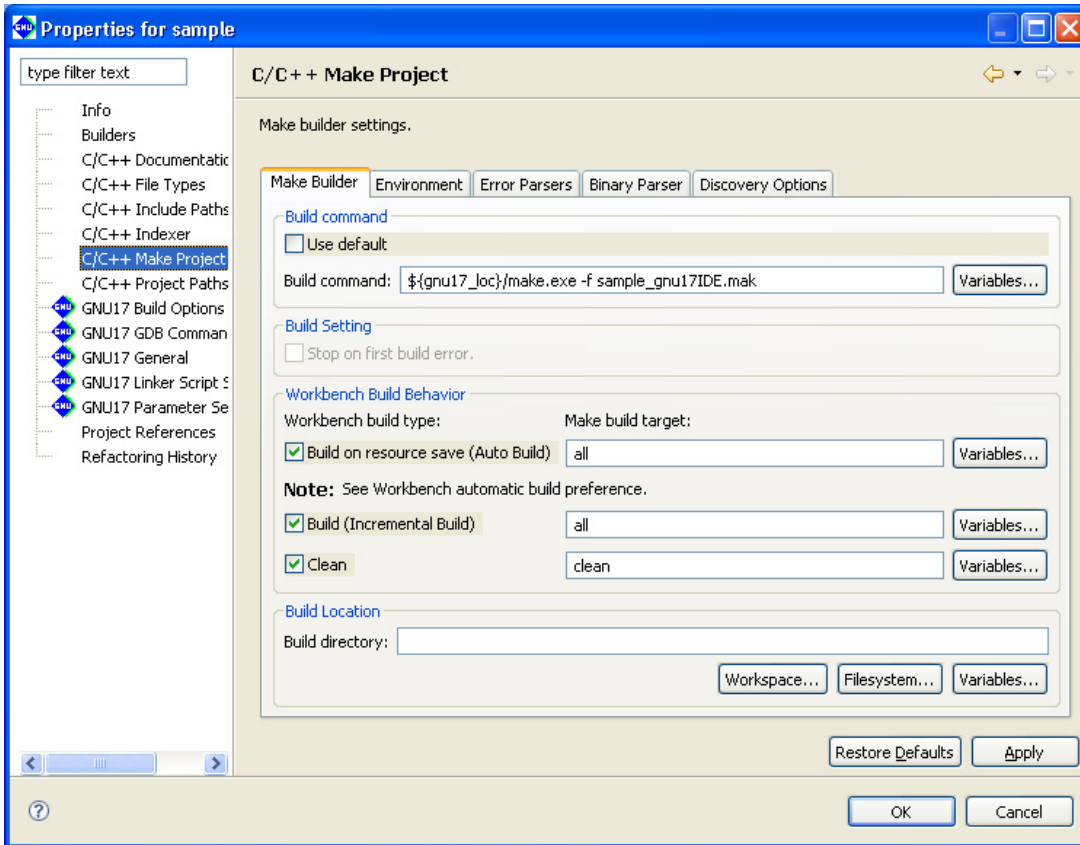
- (4) Select [Builders] from the properties list.



- (5) Deselect the [GNU17 File Builder] check box.  
The following files will no longer be generated during a build process:
  - Makefile (*<project name>\_gnu17IDE.mak*)
  - Linker script file (*<project name>\_gnu17IDE.lds*)
  - Parameter file (*<project name>\_gnu17IDE.par*)
  - Command file (*<project name>\_gnu17IDE.cmd*)

Selecting the [GNU17 File Builder] check box will cause the **IDE** to overwrite these files each time you execute a build process. If you want to use the files generated by the **IDE** except for the makefile, create a makefile with other than the name shown above and leave the [GNU17 File Builder] check box selected. Or select the [GNU17 File Builder] check box before executing a build process and deselect it after the above files have been generated.

- (6) Select [C/C++ Make Project] from the properties list to display the page for the [Make Builder] tab.



- (7) Deselect the [Use default] check box and correct the [Build command:] command line.

Example: Change to the makefile named "user.mak".

```
 ${gnu17_loc}/make.exe -f user.mak
```

- (8) Change the following target names to the ones you created. Leave the check boxes selected.

[Build (Incremental Build)]

Specify the target in the makefile called when you execute a build process. By default, "all" is called.

[Clean]

Specify the target in the makefile called when you execute a clean process. By default, "clean" is called.

There is no need to change if the same target names are used in the user makefile.

Leave the [Build on resource save (Auto Build)] check box at the default setting. (The default settings for the **IDE** disable this option.)

- (9) Click the [OK] button to close the [Properties] dialog box.

## 5.8 Starting the Debugger

Although the **gdb** debugger is provided as an application distinct from the **IDE**, it can be started from within the **IDE** after setting the appropriate command options.

### 5.8.1 Generating a Parameter File

A parameter file is used to set memory map information for the target system in the debugger. The debugger performs the following processing, based on debugger settings.

- Check to see if the software PC break addresses are within the valid map area.
- Stops program at write access to the ROM area. (simulator mode only)
- Access to undefined area (simulator mode only)
- Stack overflow (simulator mode only)

Loading a parameter file reserves sufficient storage in PC memory for all memory areas written in the file. For more information on the parameter file, refer to Section 10.8, "Parameter Files."

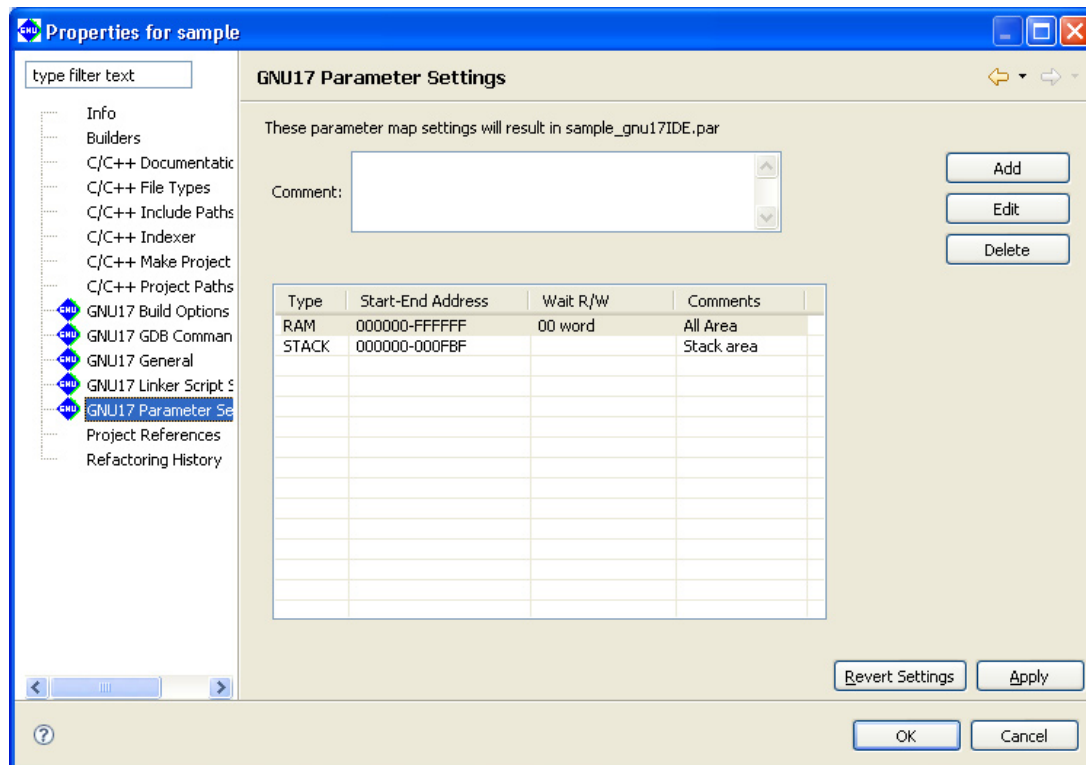
The **IDE** generates a parameter file with the name "*<project name>\_gnu17IDE.par*" during a build process, based on the project properties set. If the project properties are changed thereafter, the parameter file will be updated and passed just before you start the debugger.

The following explains how a parameter file is generated.

#### Parameter setup page

Use the [GNU17 Parameter Settings] page of the project properties to set the content of a parameter file. Display the setup page following the procedure described below.

- (1) Select a project in the [C/C++ Projects] or [Navigator] view.
- (2) Select [Properties] from the [Project] menu or the context menu of the selected project. This displays the [Properties] dialog box.
- (3) Select [GNU17 Parameter Settings] from the properties list.



This displays the currently set content in this page.

**[Comment:]**

You can enter any comments here. Up to 255 characters can be entered. The comments entered here are written in the parameter file.

**Area list**

Shows one area information in each line. Information is listed in alphabetical order, except that stack area information is displayed collectively at the bottom of the list.

**[Type]**

Shows the type of area (ROM, RAM, IO, or STACK).

**[Start-End Address]**

Shows the start and end addresses of the area in hexadecimal notation.

**[Wait R/W]**

The first two single-digit values indicates the number of wait states during a read cycle (first digit) and the number of wait states during a write cycle (second digit), respectively. The words "byte" (8 bits), "halfword" (16 bits), and "word" (32 bits) indicate the access size in which the area is accessed. If the rest is blank, the area is accessed in little endian mode. The areas set for big endian are marked by "Big".

**[Comments]**

Shows the comment entered in each area information. You do not need to enter the symbol "#" to set off comments.

Shown below are the contents initially set when you create a project.

**When S1C17 is selected for the target CPU device**

Area	Start-end address	Wait states (R/W)	Access size	Area comment
RAM	0x000000-0xFFFFFFFF	0/0	word	All Area
STACK	0x000000-0x000FBF	-	-	Stack area

**When S1C17701 is selected for the target CPU device**

Area	Start-end address	Wait states (R/W)	Access size	Area comment
RAM	0x000000-0x000FBF	0/0	word	Internal RAM
IO	0x004000-0x0043FF	0/0	byte	Peripheral Area1
IO	0x005000-0x005FFF	2/2	byte	Peripheral Area2
ROM	0x008000-0x017FFF	0/0	halfword	ROM (Flash)
RAM	0x080000-0x08055F	1/1	byte	SRAM (LCD Display)
IO	0xFFFC00-0xFFFFFFFF	0/0	byte	Reserved for Core I/O
STACK	0x000000-0x000FBF	-	-	Stack area

**Content of a parameter file****When S1C17 is selected for the target CPU device**

```
# Parameter file generated by Gnu17 Plug-in for Eclipse
```

```
RAM    000000 FFFFFFF 00W    # All Area
STACK  000000 000FBF        # Stack area
```

**When S1C17701 is selected for the target CPU device**

```
# Parameter file generated by Gnu17 Plug-in for Eclipse
ESSIM S1C17701
```

```
RAM    000000 000FBF 00W    # Internal RAM
IO     004000 0043FF 00B    # Peripheral Area1
IO     005000 005FFF 22B    # Peripheral Area2
ROM    008000 017FFF 00H    # ROM (Flash)
RAM    080000 08055F 11B    # SRAM (LCD Display)
IO     FFFC00 FFFFFFF 00B    # Reserved for Core I/O
STACK  000000 000FBF        # Stack area
```

## Editing an area

All of the above area information can be modified to suit the system. This is described below:

- (1) From the area list of the [GNU17 Parameter Settings] page, click to select the area you want to edit.
- (2) Click the [Edit] button.  
This displays the [Edit Parameter] dialog box.
- (3) Make the required settings based on the explanation given below. Click [OK].

If any address of the area you've edited overlaps that of an already set area, an error message similar to the one shown below is displayed at the top of the dialog box.

"Address range overlaps with other areas"

In such cases, correct the address of the area you're editing, or after temporarily quitting by selecting [Cancel], correct the overlapping area information before editing the information of this area once again.

- (4) Click the [Apply] button if you want to change other areas or properties or the [OK] button to end property settings.  
If you haven't clicked [Apply], you can use the [Revert Settings] button to restore modified content to the state in which this page was opened.

### [Edit Parameter] dialog box

#### [Area Type:]

Select the type of area from the following four options:

**ROM** Select to set internal or external ROM areas.

**RAM** Select to set internal or external RAM areas.

**IO** Select to map an internal I/O area or external device to memory.

**STACK** Select to set a stack area.

ROM, RAM, and IO settings are used in the debugger to determine whether software PC breakpoints are valid addresses. During simulator mode debugging, all simulated memory areas will be allocated in the PC memory. These areas cannot have overlapping addresses.

STACK settings are used specifically to cause program execution to break upon detecting a stack overflow during simulator mode and do not affect operations in any other mode or the stack pointer. Since STACK is not an area of physical memory, its addresses may overlap those of other areas. If STACK is selected, [Access Wait Cycle], [Access Size:], and [Use Big Endian] have no effect.

#### [Start Address:]

Enter the start address of the area in hexadecimal notation (omit 0x). An error will result if this address extends beyond the end address or overlaps the address of an already set area.

#### [End Address:]

Enter the end address of the area in hexadecimal notation (omit 0x). An error results if this address precedes the start address or overlaps the address of an already set area.

#### [Access Wait Cycle]

Enter the number of wait states inserted when the area is accessed. Specify in clock cycles ranging from 0 to F (hexadecimal), or 0 to 15 cycles. This is disabled for STACK areas.

[Read:] Enter the number of wait states inserted during a read cycle.

[Write:] Enter the number of wait states inserted during a write cycle.

**[Access Size:]**

Select the access size of the area from the following three options. This is disabled for STACK areas.

byte        8 bits  
 halfword   16 bits  
 word        32 bits

**[Use Big Endian]**

Select this for the area to be accessed in big endian. Unless this is selected, areas are accessed in little endian. This option is disabled for STACK areas.

**[Area Comment:]**

Enter the content of the area or other notes as a comment. You do not need to enter the symbol "#" to set off the comments.

**[OK]**

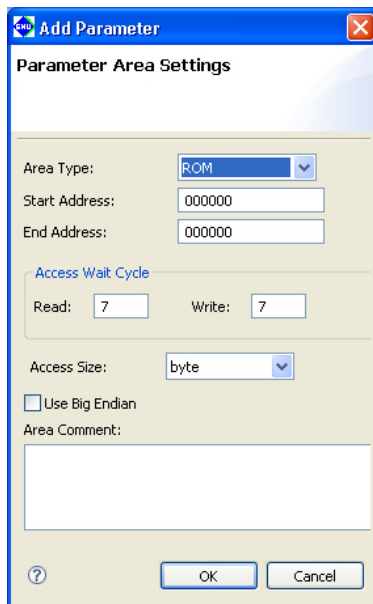
Closes the dialog box. The area list in the [GNU17 Parameter Settings] page is updated with the contents you set. If any content that needs to be set remains blank, this button is disabled. Also note that after you click [OK], the set contents are checked. If any discrepancy is detected (e.g., the set address overlaps another area), an error message is displayed at the top of the dialog box, in which case the dialog box is not closed. You must correct the erratic content of the area being edited or correct another area after temporarily quitting by selecting [Cancel].

**[Cancel]**

Discards all modifications and closes the dialog box. The area list in the [GNU17 Parameter Settings] page is not updated.

## Adding an area

Do the following to add a new area:



- (1) Click the [Add] button.

This displays the [Add Parameter] dialog box.

- (2) Make the necessary settings, based on the explanations given in "Editing an area" above. Click [OK].

- (3) Click the [Apply] button to change other areas or properties or the [OK] button to end property settings.

If you haven't clicked [Apply], you can use the [Revert Settings] button to restore modified content to the state in which this page was opened.

The area added is inserted into the list in order of set address.

## Deleting an area

Do the following to delete an unnecessary area:

- (1) From the area list of the [GNU17 Parameter Settings] page, click to select the area you want to delete.

- (2) Click the [Delete] button.

- (3) A dialog box for confirmation is displayed. Click [OK] to delete or [Cancel] to cancel.

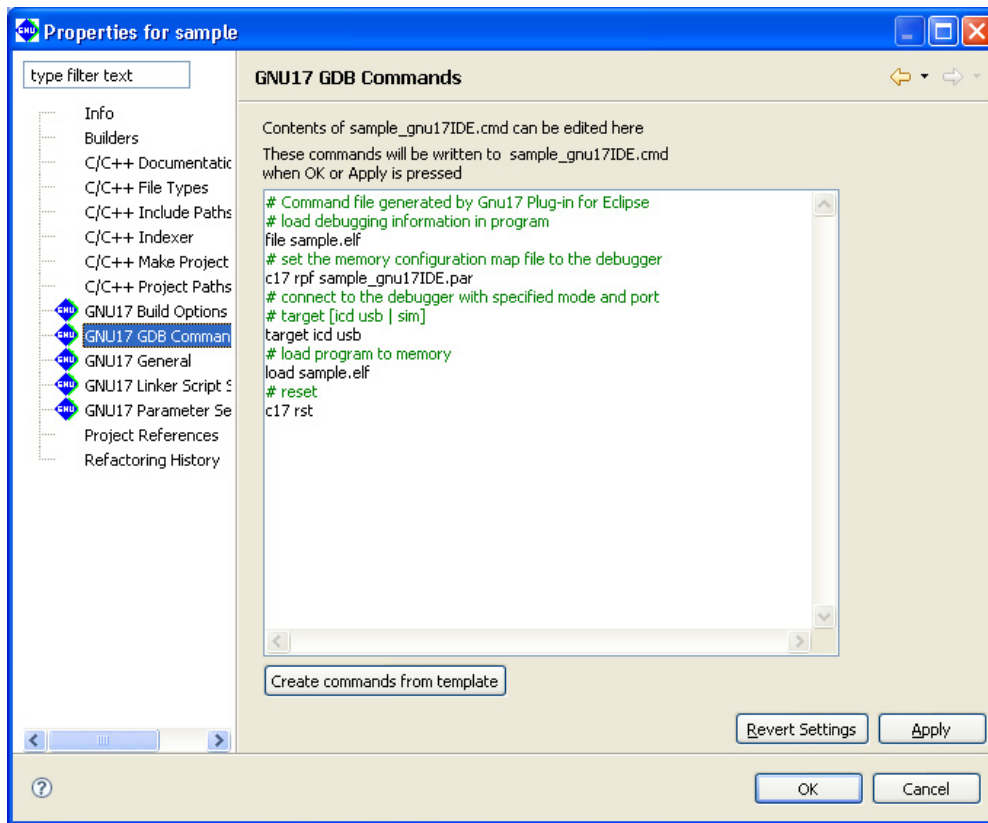
- (4) Click the [Apply] button to change other sections or properties or the [OK] button to end property settings.

If you haven't clicked [Apply], you can use the [Revert Settings] button to restore modified content to the state in which this page was opened.

## 5.8.2 Setting the Debugger Startup Commands

The debugger startup commands can be set in advance as project properties in the manner described below:

- (1) Select a project in the [C/C++ Projects] or [Navigator] view.
- (2) Select [Properties] from the [Project] menu or the context menu of the selected project.  
This displays the [Properties] dialog box.
- (3) Select [GNU17 GDB Commands] from the properties list.



This page allows the user to directly edit a debugger startup command file. The edit area shows the contents of `<project name>_gnu17IDE.cmd`. If the file does not exist, the edit area shows the default commands as below according to the settings in the project file.

- (4) Click the [Apply] button to change other sections or properties or the [OK] button to end property settings.  
If you haven't clicked [Apply], you can use the [Revert Settings] button to restore modified content to the state in which this page was opened.

The command file (`<project name>_gnu17IDE.cmd`) is generated with the contents set here when the [OK] or [Apply] button is clicked, and it will be passed to the debugger at launching.

Command file for ICD Mini mode (default)

```
# Command file generated by Gnu17 Plug-in for Eclipse
# load debugging information in program
file sample.elf           ... Loads debug information.
# set the memory configuration map file to the debugger
c17 rpf sample_gnu17IDE.par ... Loads a parameter file.
# connect to the debugger with specified mode and port
# target [icd usb | sim]
target icd usb           ... Sets connect mode.
# load program to memory
load sample.elf         ... Loads elf object file.
# reset
c17 rst                 ... Reset
```

The default setting shows the command to set the debugger in ICD Mini mode. If simulator mode is used, rewrite the command directly in this page or replace the command using the [Create a simple startup command] dialog box shown below that appears by clicking the [Create commands from template] button.

#### Command file for simulator mode

```
# Command file generated by Gnu17 Plug-in for Eclipse
# load debugging information in program
file sample.elf ... Loads debug information.
# set the memory configuration map file to the debugger
c17 rpf sample_gnu17IDE.par ... Loads a parameter file.
# set ttbr for simulator
c17 ttbr 0x008000 ... Sets TTBR.
# connect to the debugger with specified mode and port
# target [icd usb | sim]
target sim ... Sets connect mode.
# load program to memory
load sample.elf ... Loads elf object file.
# reset
c17 rst ... Reset
```

This command file is executed when the debugger starts up making the following initial settings:

1. Import debug information and parameter map information.
2. Set connect mode.
3. Set TTBR. (only in simulator mode)
4. Load the object file to be debugged.
5. Set the PC (program counter) to the boot address.

These initial settings enable the debugger to execute a program from the boot address immediately after launching.

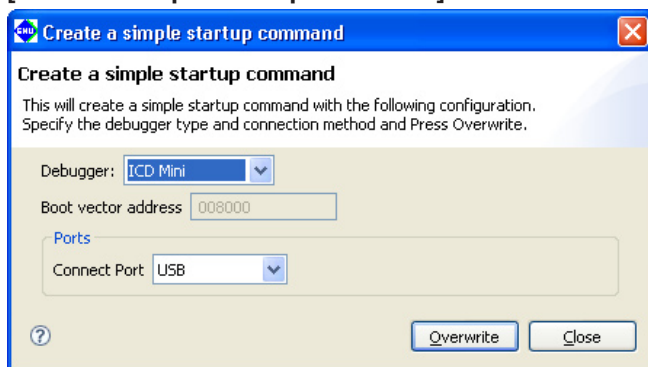
If other commands must be executed, enter them from the keyboard to add to the command list.

For detailed information on debugger commands, refer to Chapter 10, "Debugger".

The debugger startup commands in the command file may be edited using an editor.

Note, however, that the contents of the command file may not be displayed on the [GNU17 GDB Commands] page if the command file is being opened in an external editor when the [Properties] dialog box is opened. In this case, open the [Properties] dialog box after closing the command file in the external editor.

#### [Create a simple startup command]



#### [Debugger:]

Select the debugger (connect mode) to connect to.

ICD Mini When using an ICD to debug.

Simulator To debug with a PC only.

#### [Boot vector address] (effective only in simulator mode)

Enter the boot vector address in hexadecimal notation (omit 0x) in the text box when creating a command file for simulator mode. The address may be specified in 256-byte increments. The **IDE** generates a command file for simulator mode that includes a command for setting this value to the boot vector address. The value appeared here by default is the address that was specified when the project was created.



**[Ports]**

Selecting ICD Mini or Simulator in [Debugger:] sets the port to USB or NONE, respectively. So no settings are required.

**[Overwrite]**

Applies the above settings to the command file shown in the [GNU17 GDB Commands] page and closes the dialog box. A dialog box appears prompting overwrite, so execution may be canceled even after the button is clicked.

**[Close]**

Close the dialog box. The command file shown in the [GNU17 GDB Commands] page does not reflect the contents changed in the dialog box.

When you change the target CPU or memory model in the [Properties] > [GNU17 General] page, this dialog box is restored to default settings (for ICD Mini).

### 5.8.3 Starting Up the Debugger

You can start debugging when the execution file (.elf) is generated by a build operation and the preparations described in the preceding sections are complete.

#### First debugging session

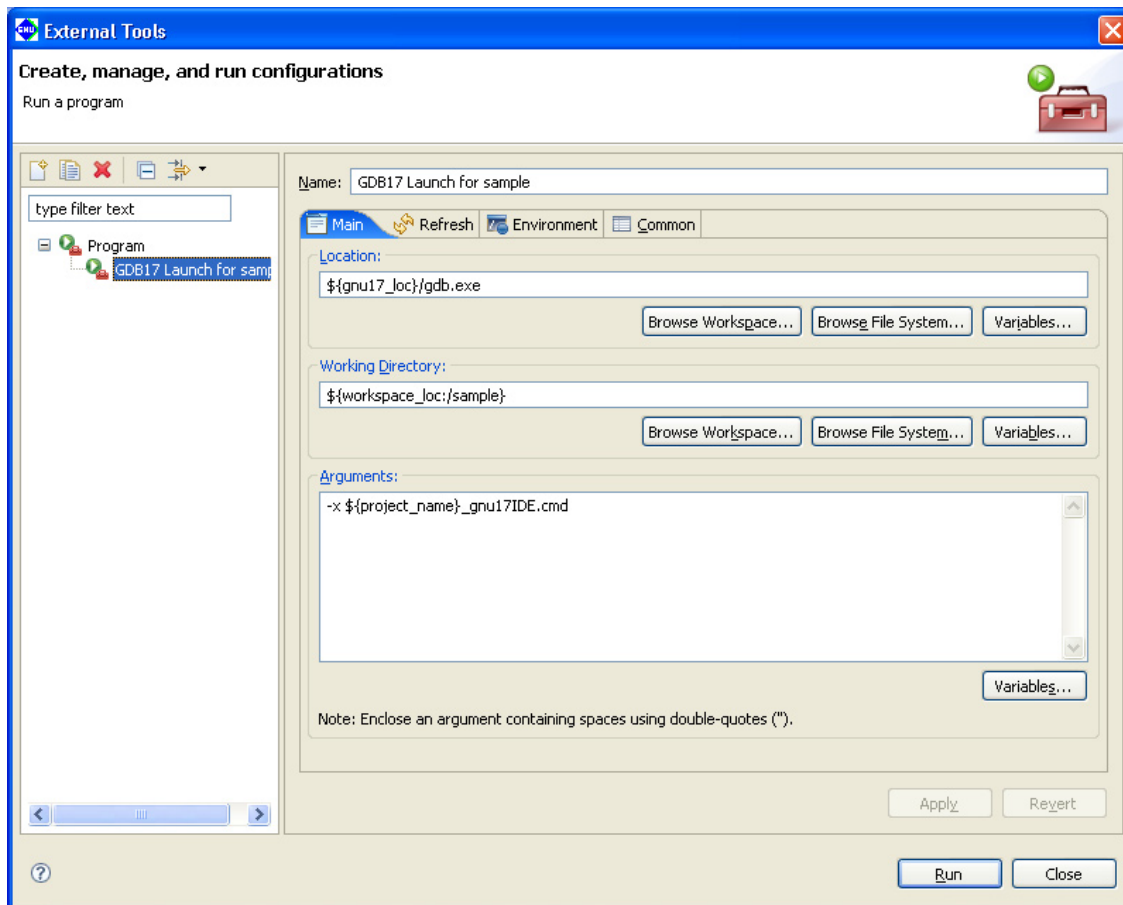
Described below is the procedure for launching the debugger for the first time for the current project.

(1) Do one of the following:

- Select [External Tools] > [External Tools...] from the [Run] menu.
- Select [External Tools...] from the [External Tools] shortcut in the window toolbar.

This displays the [External Tools] dialog box.

(2) Select [GDB17 Launch for <project name>] from the tree list.



\* `${gnu17_loc}` is resolved to the full path for `\gnu17` (e.g., `C:/EPSON/gnu17`) in the current user environment.

(3) Modify the [Arguments:] (arguments to the debugger command line) as necessary.

If you are using the **IDE**-generated command file to start the debugger, no changes are needed.

All other settings in this dialog box should also be left unchanged.

(4) Click the [Run] button.

The **gdb** debugger will launch, executing the specified command file.

For subsequent debugger operations, refer to Chapter 10, "Debugger".

## Second and subsequent debug sessions

If there is no need to change settings for the [External Tools] dialog box, do one of the following to start the second and subsequent debug sessions:

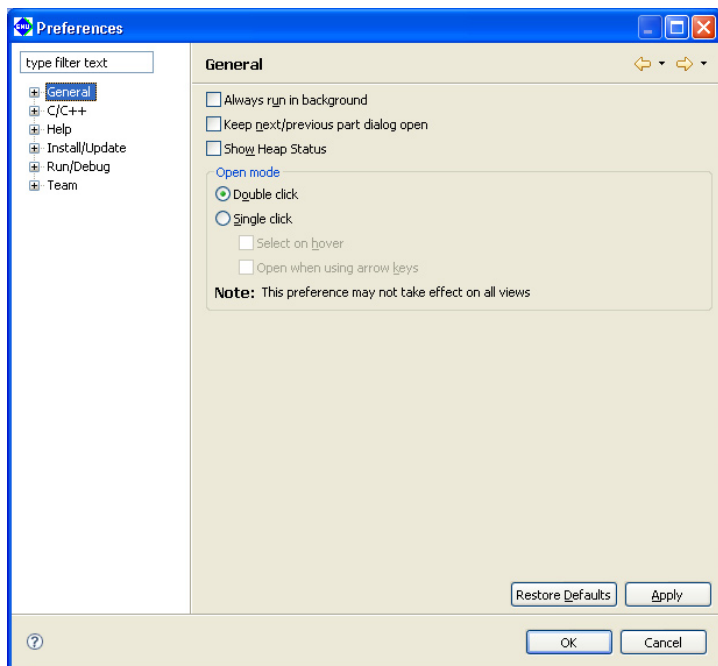
- Select [External Tools] > [GDB17 Launch for <project name>] from the [Run] menu.
- Click the [External Tools] button in the window toolbar (for the same project as in the previous debugging session).
- Select [GDB17 Launch for <project name>] from the [External Tools] shortcut in the window toolbar.

## Precautions

- Once you start the debugger, you cannot execute a build process in the **IDE**. Quit the debugger to perform a build.
- If you quit the **IDE** with the debugger open, the debugger will also be terminated.

## 5.9 Customizing the IDE (Preferences)

You can customize settings for **IDE** operations and display in various ways to suit your own needs and preferences. To perform this customization, use the [Preferences] dialog box displayed when you select [Preferences...] from the [Window] menu.



The customizable items are displayed in tree form in the left-side column of the dialog box. Select the item you want to change to display its setup page. The [type filter text] field is used to filter the items in the list so that only the items that begin with the letters to be entered will be displayed.

The buttons common to each page are described below.

 [Back]

Returns to the preference page previously referenced or edited.

 [Forward]

Reverts the display traced back by [Back] above to the next recent page.

[Restore Defaults]

Restores the set content of each page to the state in which the dialog box was opened or the state at which the [Apply] button was clicked to confirm dialog box settings.

[Apply]

Applies the settings made on the page. Before changing another item, be sure to click [Apply] before proceeding to a new page.

[OK]

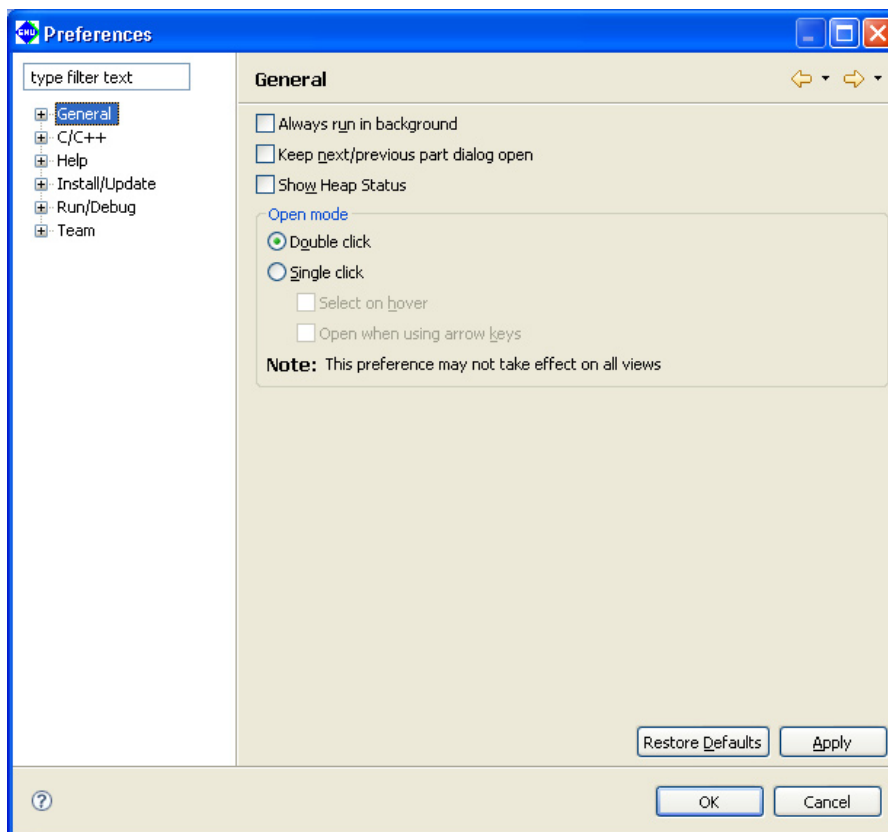
Applies the set content of the current page and closes the dialog box.

[Cancel]

Cancels settings and closes the dialog box. Settings already confirmed with the [Apply] button prior to [Cancel] will not be canceled.

The relevant page and the set content of each customization item are described below. Please do not attempt to change any settings not discussed here.

## General



Make settings for **IDE** operations.

[Always run in background] (default: OFF)

If this check box is selected, a build or other process runs in the background (no dialogs displayed during a build), allowing you to perform other tasks.

[Keep next/previous part dialog open] (default: OFF)

By pressing [Ctrl] + [F6] (for editor) or [Ctrl] + [F7] (for views), the **IDE** switches the editor/view between one currently edited/referenced and another one previously edited/referenced. Normally, pressing the keys switches the editor/view immediately. If this check box is selected, pressing the keys displays a pull-down menu including the browsing history and you can select the editor/view to be activated from the list.

[Show Heap Status] (default: OFF)

If this check box is selected, the usage status of the Java heap will be displayed at the bottom right of the workbench window.

[Open mode]

Select the action by which resources are opened in the editor from the [C/C++ Projects] or [Navigator] view.

[Double click] (default: ON)

Single-clicking a resource selects it; double-clicking a resource opens it.

[Single click] (default: OFF)

Single-clicking a resource opens it.

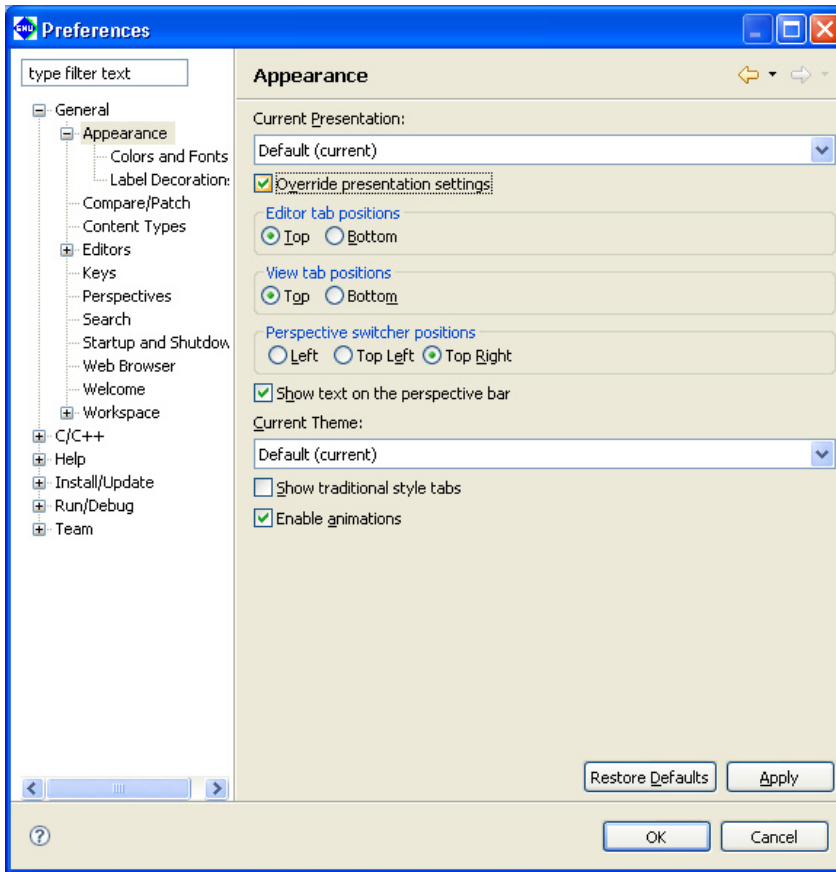
[Select on hover] (default: OFF)

If this check box is selected, a resource in the view can be selected simply by hovering the mouse cursor over it. (Effective only when [Single click] is selected)

[Open when using arrow keys] (default: OFF)

If this check box is selected, a resource can also be opened in the editor by selecting it with arrow keys. (Effective only when [Single click] is selected)

## General &gt; Appearance



Specify the **IDE** appearance and the tab positions displayed in the editor and views.

[Current Presentation:]

The appearance of the **IDE** can be changed to the Eclipse 2.1 style. To switch the appearance, restart the **IDE** after selecting it from the pull-down list.

[Override presentation settings] (default: OFF)

Select this check box when changing the tab position displayed in the editor.

[Editor tab positions]

Specify the tab position displayed in the editor.

[Top] (default: ON)

Tabs are displayed at the top of the view.

[Bottom] (default: OFF)

Tabs are displayed at the bottom of the view.

[View tab positions]

Specify the tab position displayed in the view.

[Top] (default: ON)

Tabs are displayed at the top of the view.

[Bottom] (default: OFF)

Tabs are displayed at the bottom of the view.

**[Perspective switcher positions]**

Specify the displayed position of the perspective bar.

**[Left]** (default: OFF)

The perspective bar is displayed on the left edge of the window.

**[Top Left]** (default: OFF)

The perspective bar is displayed at the upper left part of the window (below the toolbar).

**[Top Right]** (default: ON)

The perspective bar is displayed at the upper right part of the window (on the right of the toolbar).

**[Show text on the perspective bar]** (default: ON)

If this check box is selected, a perspective name is also displayed on the perspective bar. If this check box is unselected, only the icon is displayed.

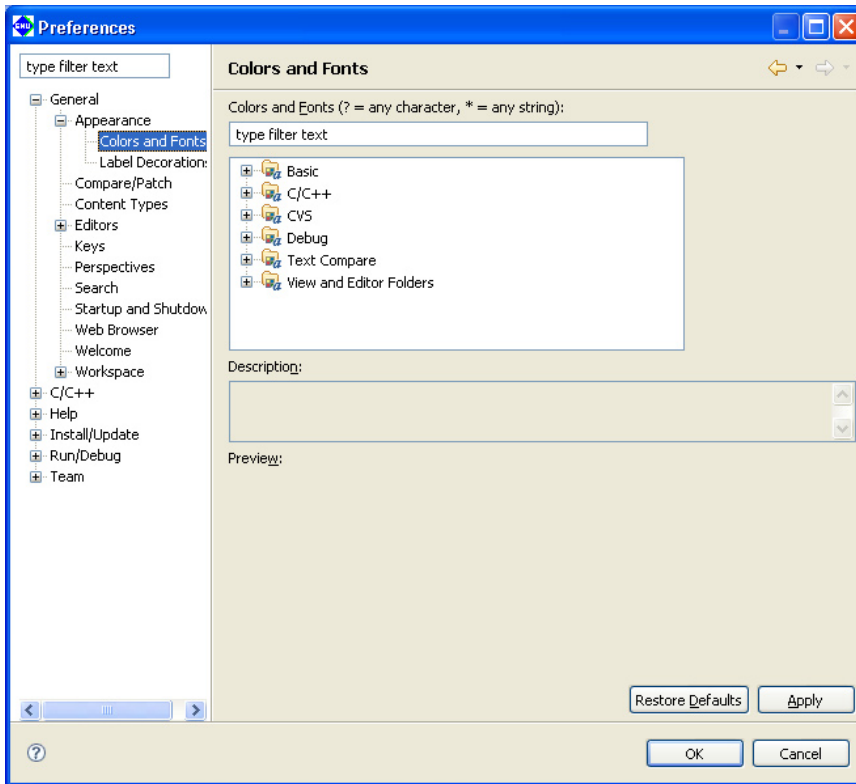
**[Show traditional style tabs]** (default: OFF)

Selecting this check box displays the tabs in the editor and views in an angular style.

**[Enable animations]** (default: ON)

Selecting this check box enables the function to animate fast views to their location when they are closed or opened.

## General &gt; Appearance &gt; Colors and Fonts



Set the fonts and colors used in the editor and other windows.

[Colors and Fonts:]

Specify the items to be displayed in the list. Use the symbols "?" and "\*" as wildcards to specify any character or any string, respectively.

List box

Color and font settings are listed by category here. Select the color or font you want to change from this list.

[Description:]

Displays a description of the location, etc. in or for which the color or font you selected from the list will be used.

[Preview:]

If available, a display sample of the color or font selected from the list is displayed here.

The buttons described below are displayed when you select a font from the list.

[Use System Font]

Changes the font you selected from the list to the system font.

[Change...]

Changes the font selected from the list to a font or font size selected in the [Font] dialog box.

[Reset]

Restore the changed font to the default font. This button is enabled once a font is changed.

The buttons described below are displayed when you select a color from the list.

Color select button

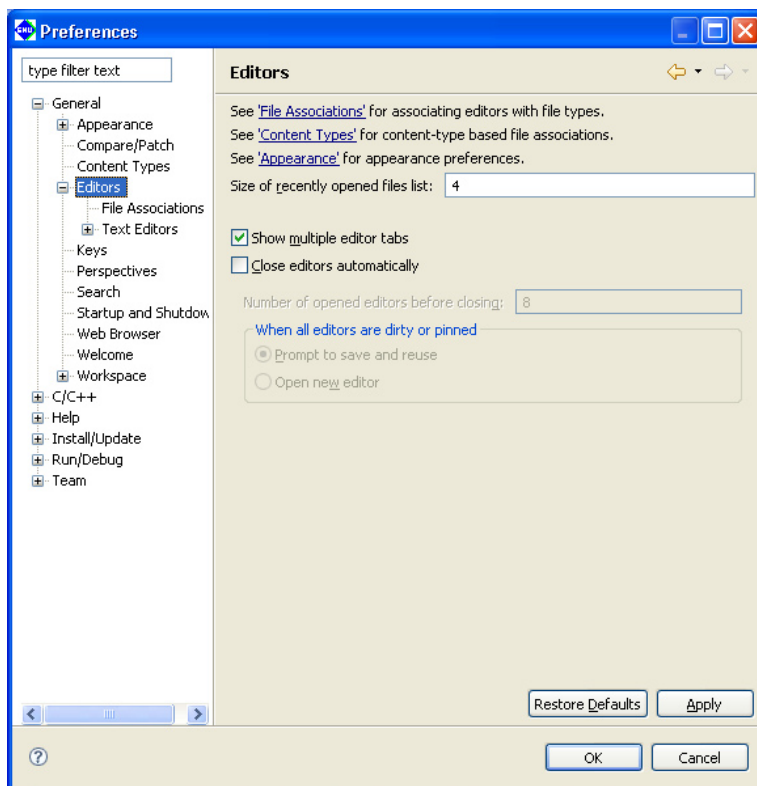
Select a new color in the [Color] dialog box.

[Reset]

Resets the changed color to the default color. This button is enabled once a color is changed.



## General &gt; Editors



Set the items associated with all editing windows.

[Size of recently opened files list:] (default: four)

Set the number of recently opened files displayed on the [File] menu.

[Show multiple editor tabs] (default: ON)

Choose whether to display multiple tabs at the same time in the editor view.

If you deselect this check box, only the tab for the foremost document is displayed. In this case, use the shortcut menu (>>) located above the tab to select other documents.

[Close editors automatically] (default: OFF)

If more than a specified number of editors is open (by default, 8 editors), selecting this check box automatically closes the editors, starting with the oldest.

If this feature is selected, the [Pin Editor] button appears in the toolbar. Selecting this button (leaving it depressed) will leave the resource open when others are closed automatically. (You can do the same by selecting [Pin Editor] from the context menu of the tab.)

If any file to be closed remains unsaved, a dialog box prompts you to save or discard changes or to choose to open an editor that would exceed the limited number of editors.

[Number of opened editors before closing:] (default: eight)

Specify the number of resources opened in the editor at which you want the above auto-close feature to be enabled. This field is settable when [Close editors automatically] is selected.

[When all editors are dirty or pinned]

When the auto-close feature is on, specify the processing to be performed when a limited number of editors is already open and not all are saved or the "Pin Editor" is selected. This field is settable when [Close editors automatically] is selected.

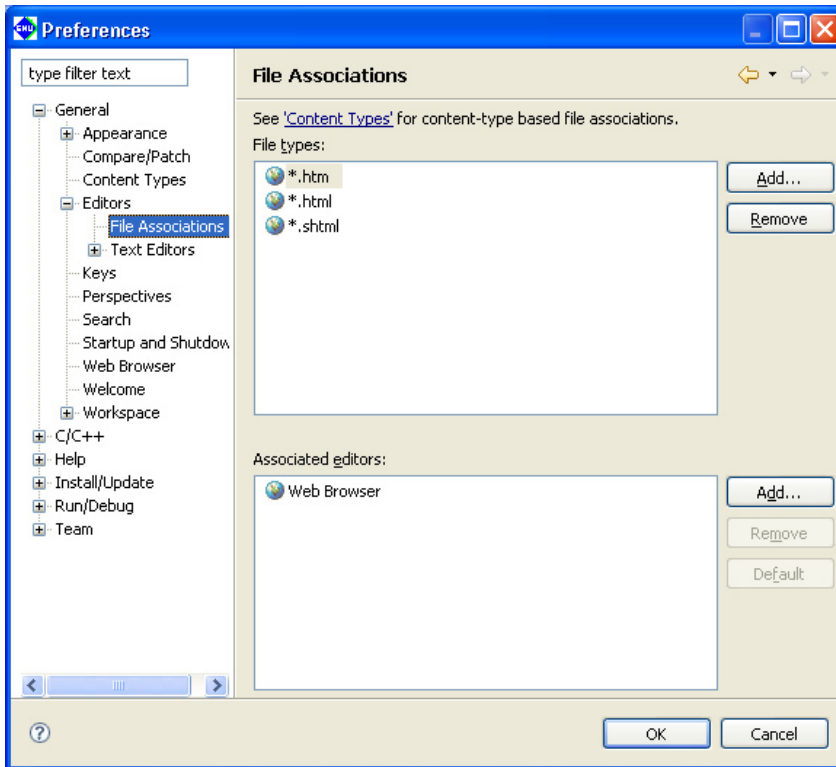
[Prompt to save and reuse] (default: ON)

Shows a dialog box that allows you to choose to save or not save the file to be closed, or choose to newly an editor surpassing the limited number of editors.

[Open new editor] (default: OFF)

Opens an editor surpassing the limited number of editors without asking for your confirmation.

## General &gt; Editors &gt; File Associations



Set a file type (file name extension) and the editor used to edit it.

[File types:]

Lists the file types to be edited in the **IDE**.

[Associated editors:]

Lists the editors used to edit the files selected in [File types:]. The editor indicated as the default in parentheses is used if you open a file by double-clicking in the [C/C++ Projects] or [Navigator] view. You can use another editor on the list by selecting the [Open With] command from the context menu.

[Add...]

Adds a file type or editor to the list. You can enter or select one in the dialog box that appears when you click this button.

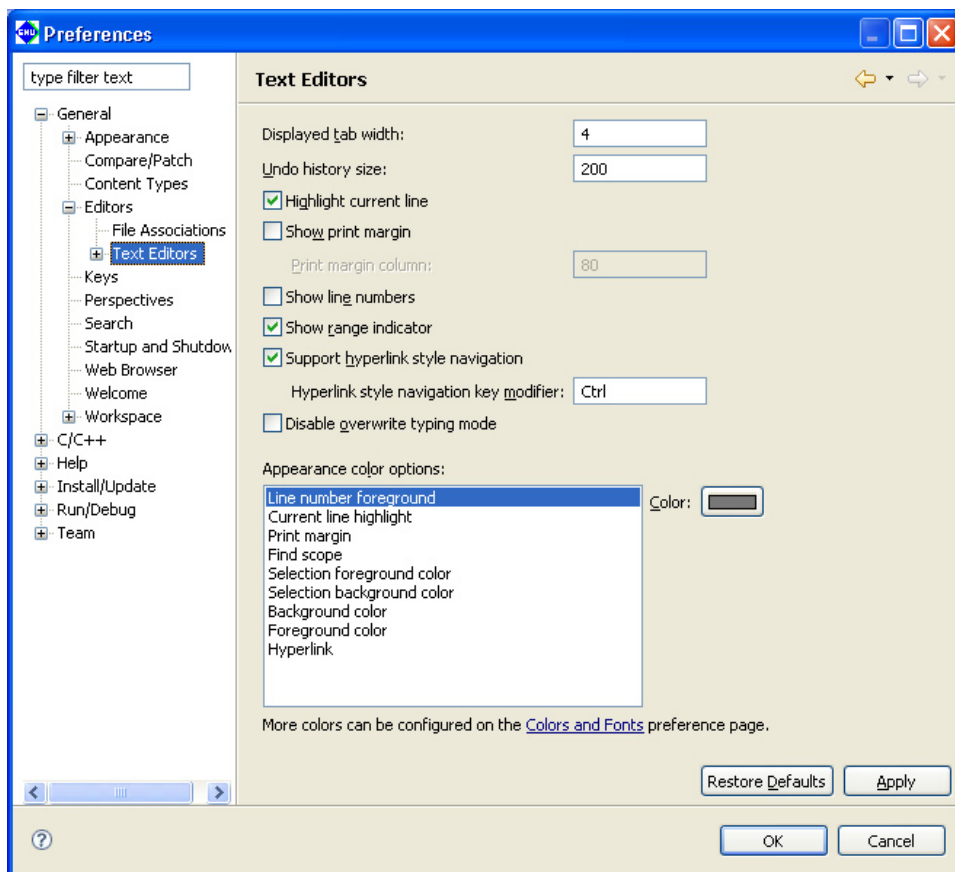
[Remove]

Removes a selected file type or editor from the list box.

[Default]

Sets the editor selected in [Associated editors:] to the default editor.

## General &gt; Editors &gt; Text Editors



Make settings for the IDE's text editor.

[Displayed tab width:] (default: four)

Specify the tab width in number of characters.

[Undo history size:] (default: 200)

Specify the number of times to undo the recent operations performed.

[Highlight current line] (default: ON)

If this check box is selected, the current line is highlighted while at the same time tinted with a color.

[Show print margin] (default: OFF)

If this check box is selected, a vertical line is displayed to indicate a print margin (per-line printable range set by [Print margin column:]).

[Print margin column:] (default: 80)

Specify the number of characters printed per line.

[Show line numbers] (default: OFF)

If this check box is selected, a line number is displayed at the beginning of each line.

[Show range indicator] (default: ON)

If this check box is selected, the marker bar at the left edge of the editor area will display the range indicator that shows the location of the function or other element being selected in the [Outline] view.

[Support hyperlink style navigation] (default: ON)

Selecting this check box enables the hyperlink style navigation feature allowing you to jump to the location where the function/variable is declared by clicking its name by holding down the [Ctrl] key (default).

[Hyperlink style navigation key modifier:] (default: [Ctrl] key)

Select a key used for hyperlink style navigation.

[Disable overwrite typing mode] (default: OFF)

Selecting this check box disables overwrite typing mode.

[Appearance color options:]

Set the following display colors. Select the item whose display color you want to change from the list, then select a color from the dialog box displayed when you click the [Color:] button.

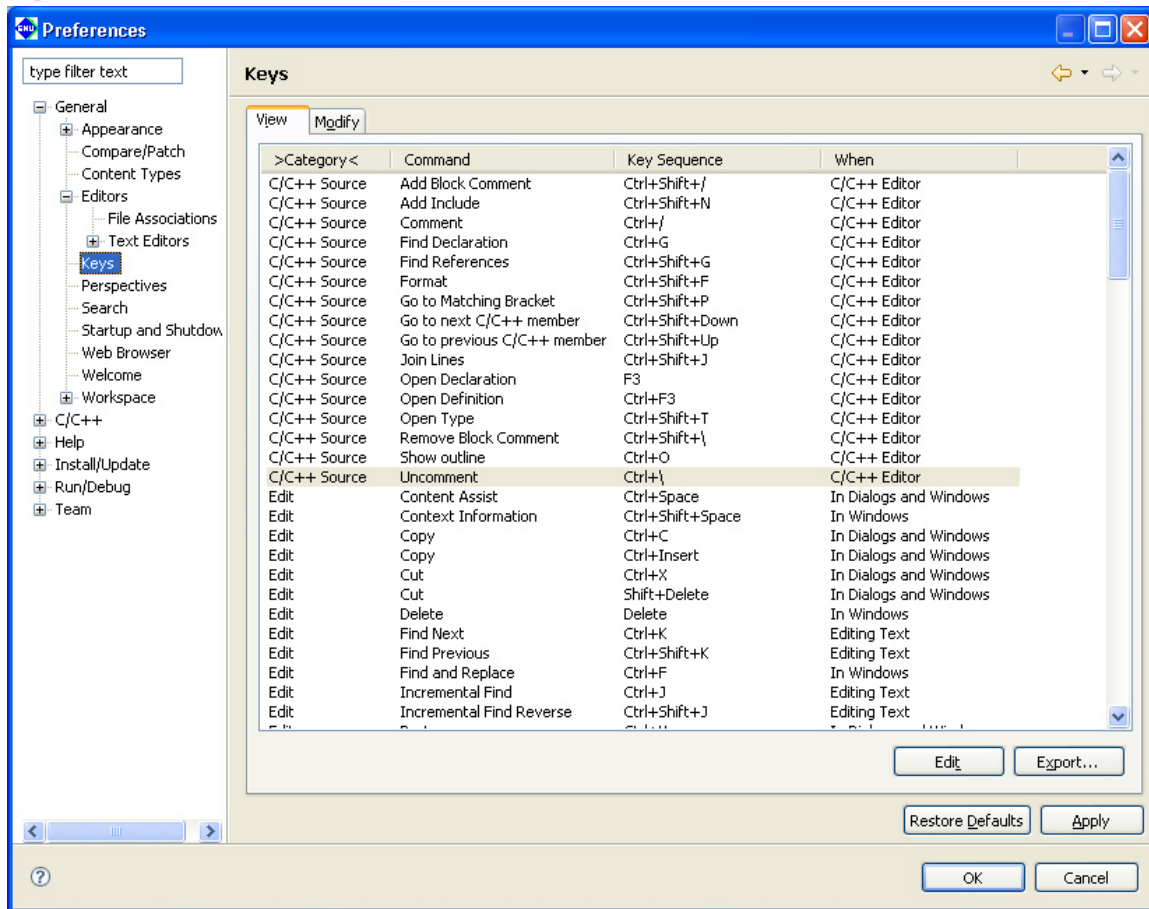
Line number foreground	Line number character color
Current line highlight	Current line highlight color
Print margin	Vertical line color showing a print margin
Find scope	Range of search area
Selection foreground color	Character color* of a selection
Selection background color	Background color* of a selection
Background color	Background color*
Foreground color	Character color*
Hyperlink	Hyper link character color

\* If the [System Default] check box is selected, default settings of the system are applied.

## General > Keys

Set keyboard shortcuts.

### [View] tab



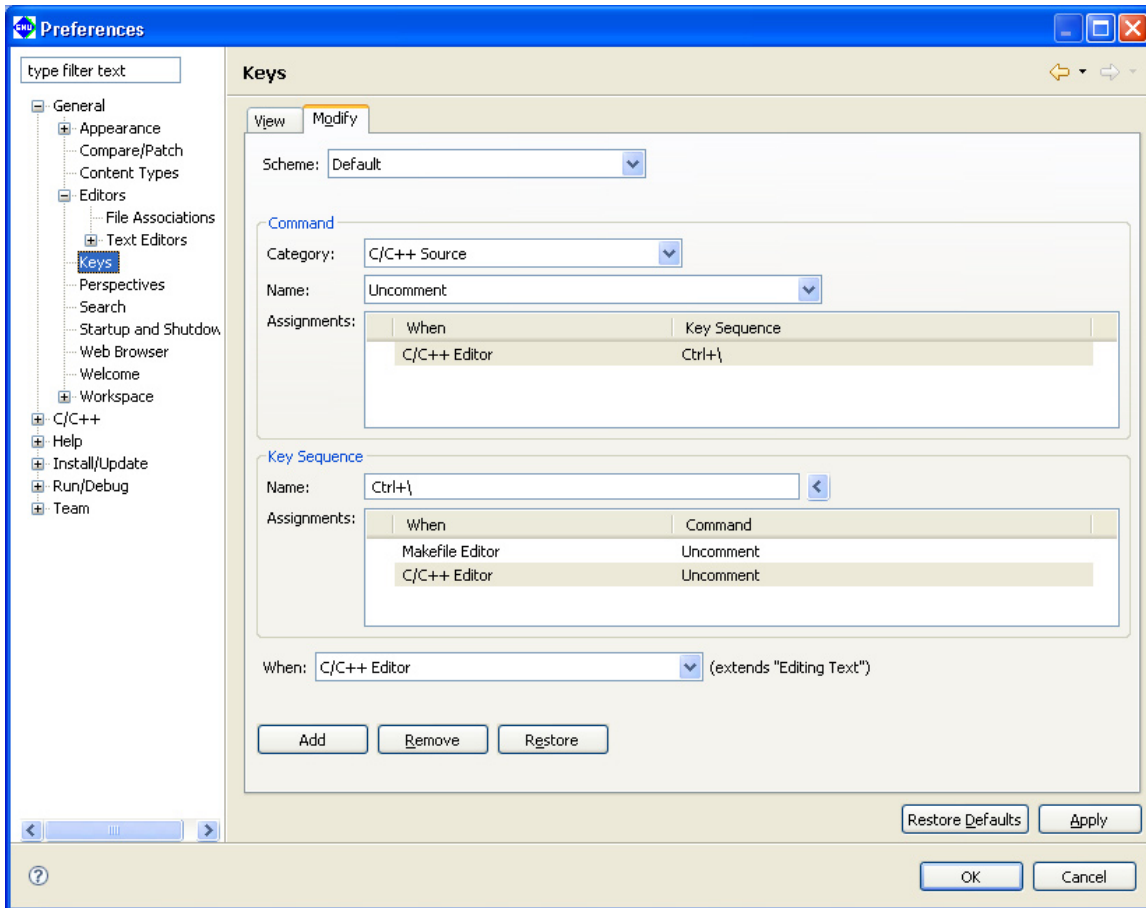
Shows the list of shortcut keys.

### [Edit]

Opens the [Modify] tab page to change the key sequence for the command selected in the list.

### [Export...]

Saves the contents of the list to a CSV format file.

**[Modify] tab**

Add, edit, and remove shortcut keys.

**[Scheme:]**

The **IDE** supports two sets of keyboard shortcuts, either of which is selectable. The "Default" set is a keyboard shortcut normally set in Windows. The "Emacs" set is an Emacs-compliant extended version of the "Default" keyboard shortcut set.

**[Command]**

Select the command here to reference, assign or delete a key sequence.

**[Category:]**

Select a command category.

**[Name:]**

Select a command name.

**[Assignments:]**

Shows the location/context in which a command is valid and the currently assigned key sequence.

**[Key Sequence]**

Here, set the key sequence to be assigned to a command. Double-click a keystroke displayed in the [Assignments:] column of the [Command] section to review all commands currently assigned to that key sequence.

**[Name:]**

Enter a key sequence. Holding down the [Ctrl] key while pressing the [A] key (for example) registers the "Ctrl + A" keystroke combination. To configure a key sequence with multiple keystrokes, enter the actual keystrokes in the desired order. To enter a tab stop and backspace as a keystroke sequence, select from the list displayed by clicking the button located to the right of [Name:].

**[Assignments:]**

Shows the command assigned to the key sequence registered in [Name:] and the location/context in which the command is valid. Before assigning a key sequence to a command, check for any other command enabled in the same location/context. Use the [Remove] button to remove any duplicated commands.

**[When:]**

When assigning the key sequence set in [Key Sequence] to the command selected in [Command], select the location/context in which the command is enabled.

**[Add]**

Assigns the key sequence set in [Key Sequence] to the command selected in [Command], enabling this correspondence for the location/context selected in [When:].

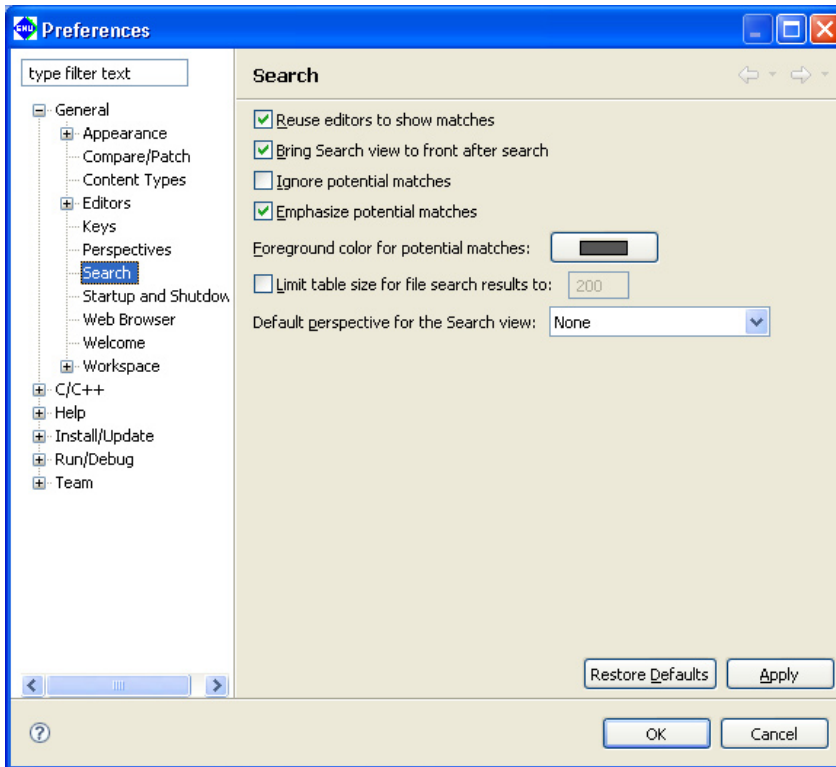
**[Remove]**

Removes the key sequence or command selected in [Assignments:].

**[Restore]**

Discards any changes made and restores the default settings.

## General &gt; Search



Make settings related to searches performed with the [Search] menu/button.

[Reuse editors to show matches] (default: ON)

If this check box is selected, the same editor is used to show search results. The current file is closed if you move to a search position in another file that can be displayed in the same editor.

[Bring Search view to front after search] (default: ON)

If this check box is selected, the [Search] view is displayed in front of other views after a search.

[Ignore potential matches] (default: OFF)

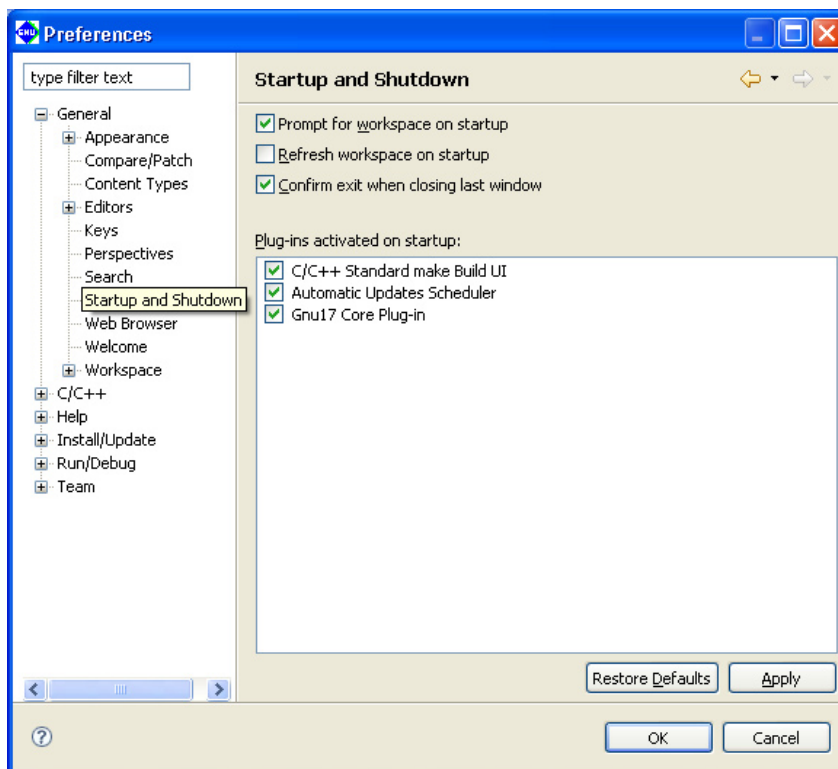
If this check box is selected, only complete matches for the search text are displayed.

[Limit table size for file search results to:] (default: OFF)

Select this check box to limit the number of found occurrences of search text and to set the number of occurrences displayed.



## General &gt; Startup and Shutdown



Make settings related to **IDE** startup/shutdown.


[Prompt for workspace on startup] (default: ON)

If this check box is selected, a dialog box for specifying a workspace directory is displayed on **IDE** startup.

[Refresh workspace on startup] (default: OFF)

If this check box is selected, workspace information is updated to the latest file system status on **IDE** startup.

[Confirm exit when closing last window] (default: ON)

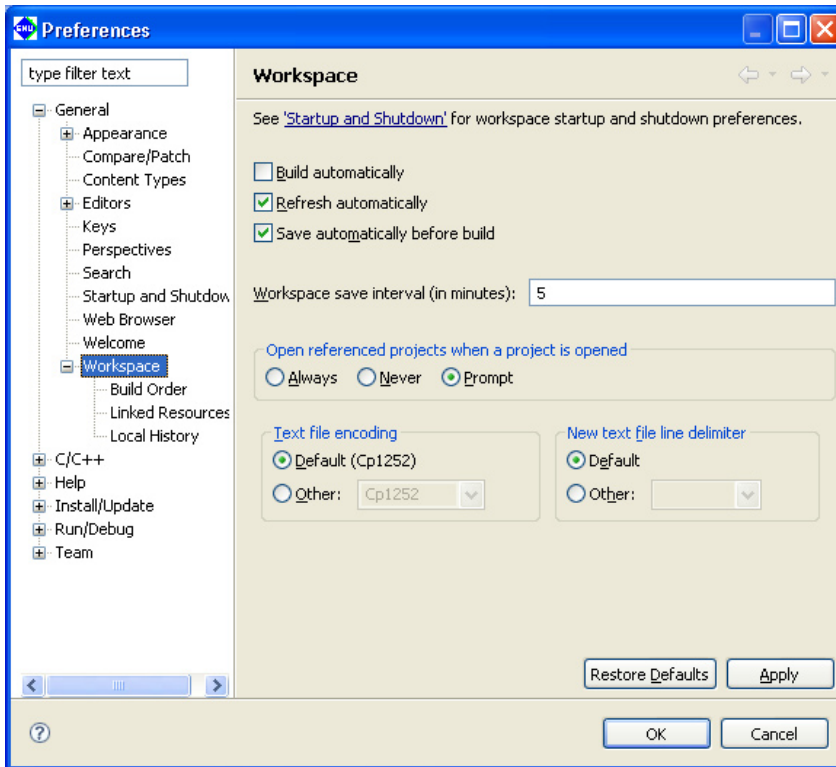
If this check box is selected, a dialog box prompting for confirmation is displayed if you click the  (Close) button to close the last open **IDE** window.

[Plug-ins activated on startup:]

Select plug-ins you want to activate on **IDE** startup.

Do not modify this setting.

## General &gt; Workspace



Make settings for **IDE** operations.

[Build automatically] (default: OFF)

Although this check box is provided to enable or disable the auto-build feature (automatically build a project when you save sources you've been editing in the editor), this feature cannot be used in the **IDE**.

[Refresh automatically] (default: ON)

If this check box is selected, resources in the file system are automatically reflected in the [C/C++ Projects] and [Navigator] views when added or removed. When this check box is deselected, select [Refresh] from the [File] menu or the context menu of the view to update the display of the view.

[Save automatically before build] (default: ON)

If this check box is selected, the resources being edited in the editor but not yet saved will be automatically saved before a build process is executed.

[Workspace save interval (in minutes):] (default: five minutes)

Set the intervals in minutes at which intervals you want the workspace information to be automatically saved.

[Open referenced projects when a project is opened]

Select whether opening a project will also open other projects it references or not.

[Always] (default: OFF)

Referenced projects will always be opened.

[Never] (default: OFF)

No other projects will be opened.

[Prompt] (default: ON)

A dialog appears to prompt for selection.

**[Text file encoding]**

Set text encoding format.

**[Default (\*1)]** (\*1: Cp1252, MS932, etc.; varies with Windows language support)

This is the standard Windows character set. (default)

**[Other:]**

Select another character encoding.

**[New text file line delimiter]**

Select a line delimiter. The selection will take effect for subsequent only. It does not affect the existing files.

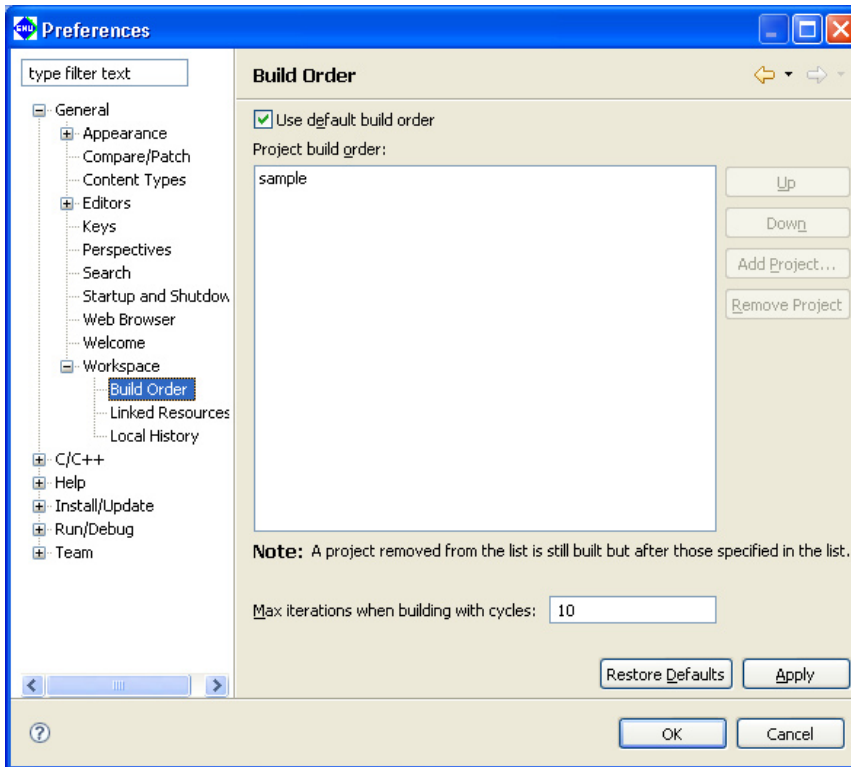
**[Default]**

The standard Windows line delimiter is used. (default)

**[Other:]**

Select another line delimiter.

## General &gt; Workspace &gt; Build Order



Define the order in which projects are built.

[Use default build order] (default: ON)

If [Build All] (building all the opened projects) is executed when this option is selected, the projects are built in the order in which they appear in the [C/C++ Projects] view (in alphabetical order). If this check box is deselected, the projects will be built in the order of the list on this page.

[Project build order:]

Set the build order in this field when the [Use default build order] check box is deselected.

The projects will be built from the top of the list. If projects not listed here are opened, they will be built in alphabetical order after all projects in this list are built.

[Up]

Moves the project selected one position up in the list.

[Down]

Moves the project selected one position down in the list.

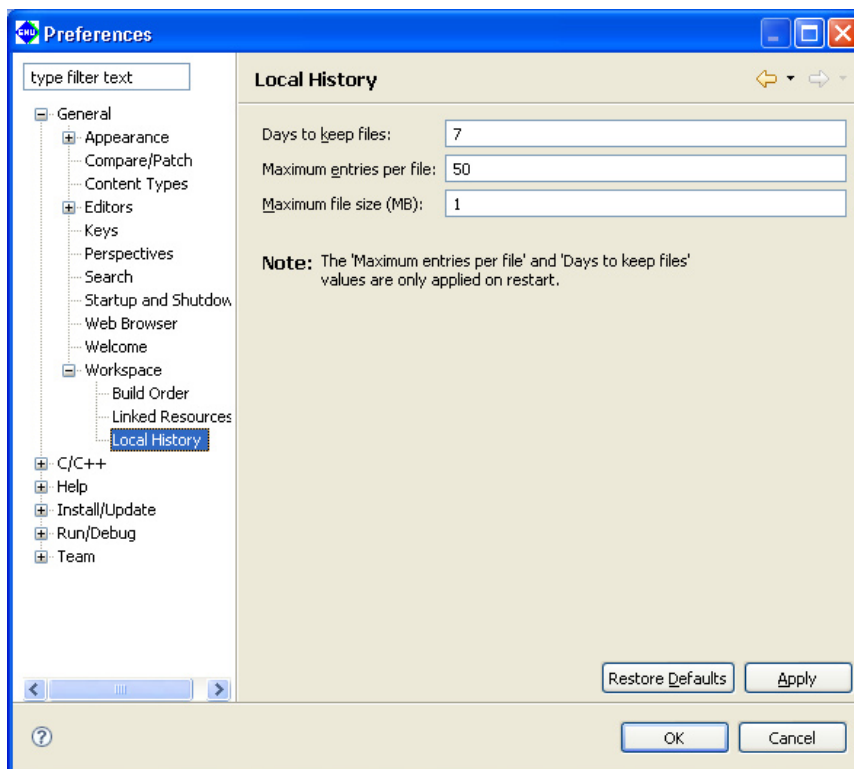
[Add Project...]

Adds a project in the list.

[Max iterations when building with cycles:] (default: 10)

Set the maximum number of times to build the projects in the order specified in the list.

## General &gt; Workspace &gt; Local History



Make settings related to the history of modified resources.

[Days to keep files:] (default: seven days)

Set the number of days you want the revision history to be retained.

[Maximum entries per file:] (default: 50)

Set the number of entries of revision history retained per file.

[Maximum file size (MB):] (default: 1MB)

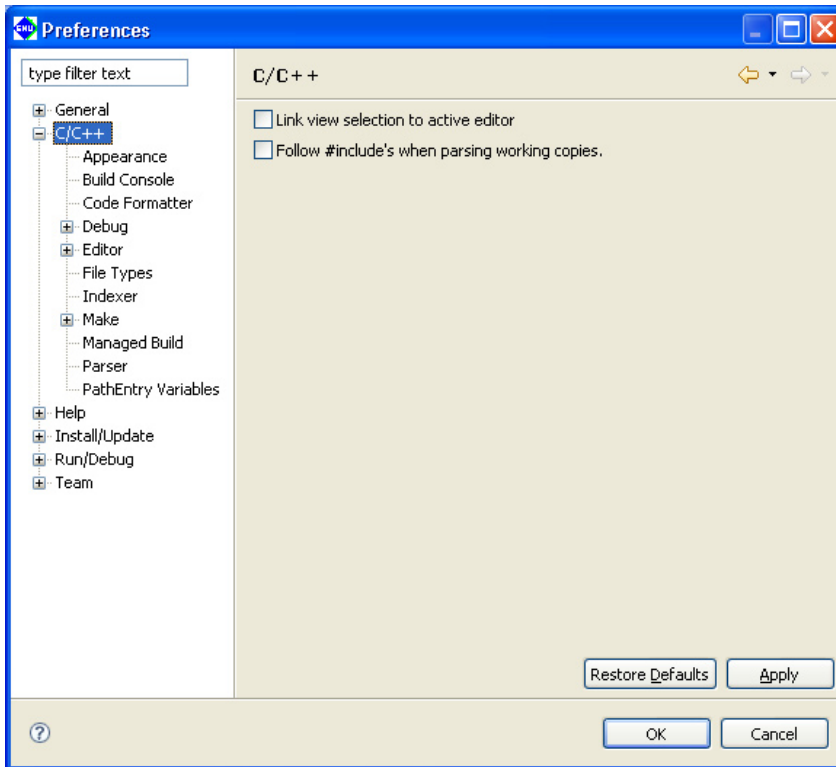
Set the maximum file size for the file containing revision history entries.

No history entry will be saved for a file whose size exceed the value set here.

History entries exceeding these limits will be erased.

Restart the **IDE** to apply these settings.

## C/C++



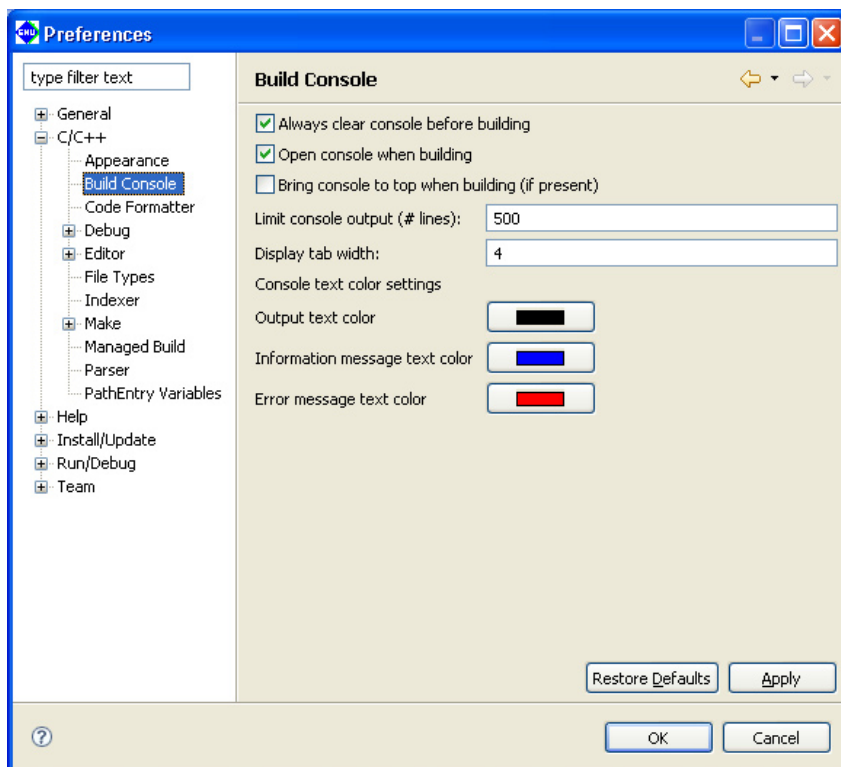
Make settings related to the [C/C++ Projects] view.

[Link view selection to active editor] (default: OFF)

Selecting this check box enables the [C/C++ Projects] view and the editor linkage feature. When linkage is enabled, the resource selected in the [C/C++ Projects] view is displayed in front of all other resources in the editor (assuming the editor is already open). This setting does the same thing as enabling (selecting by pressing) the [Link with Editor] button in the [C/C++ Projects] view. Deselecting the [Link with Editor] button also switches off this check box.

This setting is not interlocked with the [Link with Editor] button in the [Navigator] view.

## C/C++ &gt; Build Console



Make settings related to the [Console] view.

[Always clear console before building] (default: ON)

Selecting this check box clears [Console] view when you begin building a project.

[Open console when building] (default: ON)

Selecting this check box opens the [Console] view when you build a project.

[Bring console to top when building (if present)] (default: OFF)

Selecting this check box displays the [Console] view in front of other views (if already open) when you build a project.

[Limit console output (# lines):] (default: 500 lines)

Specify the maximum number of lines displayed in the [Console] view.

[Display tab width:] (default: four characters)

Specify a number of characters for the tab width displayed in the [Console] view.

[Console text color settings]

[Output text color]

This is the display color for executed command lines.

[Information message text color]

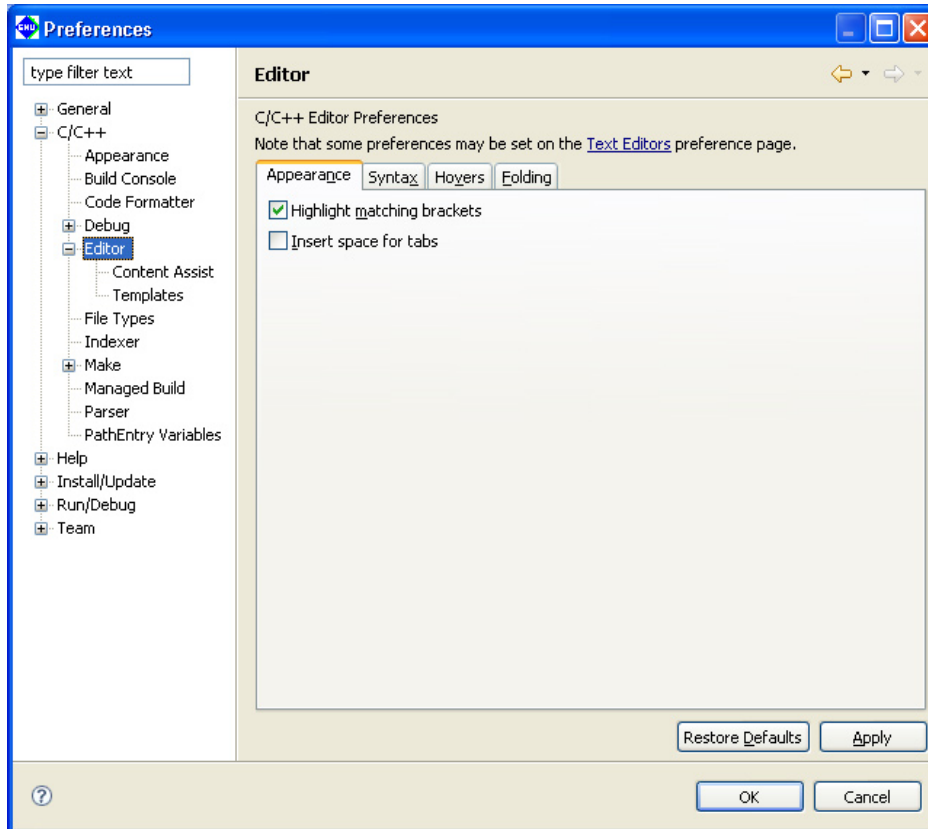
This is the display color for status messages output by tools.

[Error message text color]

This is the display color for error messages output by tools.

**C/C++ > Editor**

Make settings related to the C editor incorporated in the **IDE**.

**[Appearance] tab**

Make display settings.

**[Highlight matching brackets]** (default: ON)

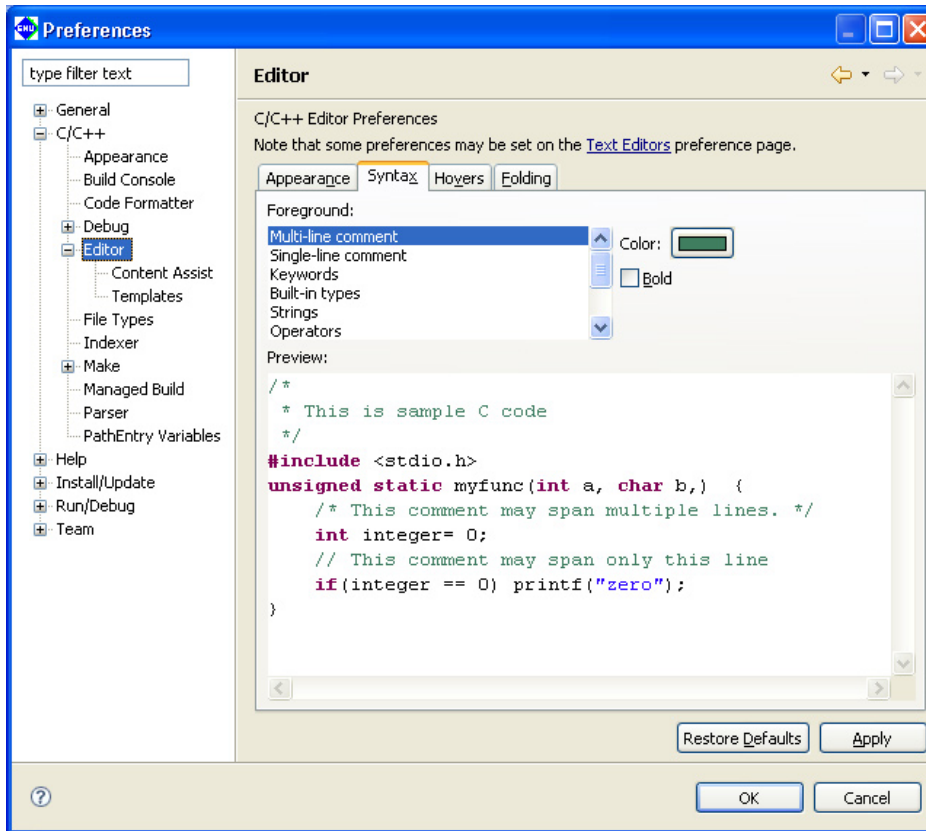
Selecting this check box highlights the corresponding brace ('{' or '}') with a bounding rectangle when you place the cursor on a brace.

**[Insert space for tabs]** (default: OFF)

Selecting this check box replaces [Tab] keys with spaces equal to the corresponding number of characters corresponding to the set Tab interval.



## [Syntax] tab



Set colors.

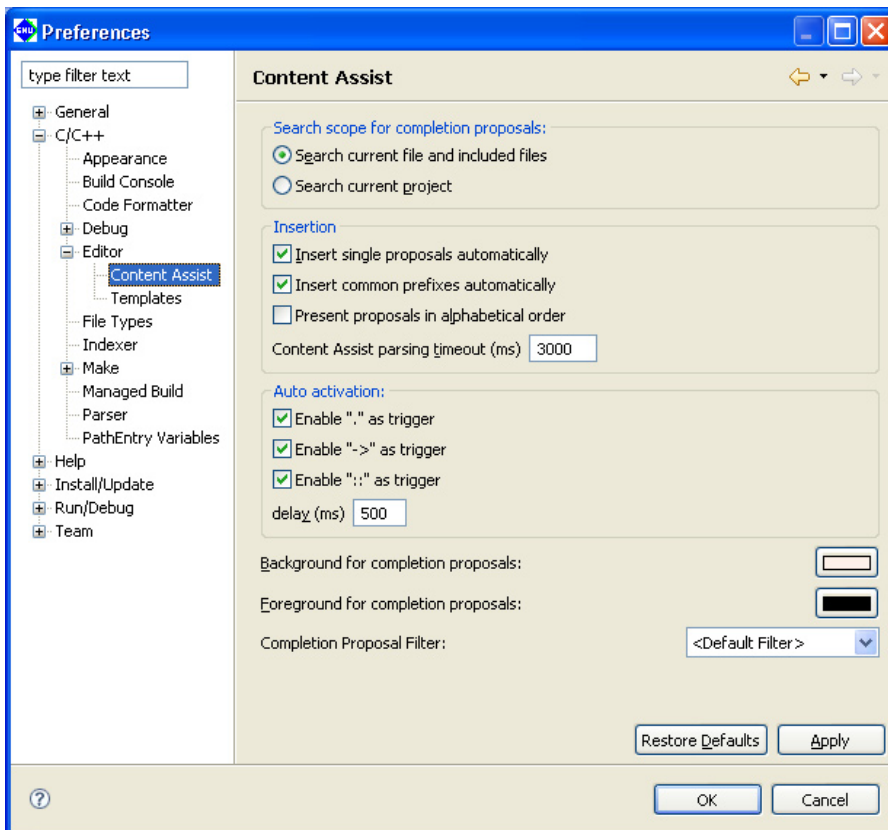
## [Foreground:]

Set a text color for comments and keywords. Select a statement type from the list, click the button located to the right, and select the desired color in the ensuing [Color] dialog box. Select the [Bold] check box to display with a bold font.

## [Preview:]

Shows a sample of the settings made above for your confirmation.

## C/C++ &gt; Editor &gt; Content Assist



Make settings related to content assist.  
Settings should usually be left on default.

## [Search scope for completion proposals:]

Set the range of potential candidates for selection in a content assist list, other than reserved words and templates.

[Search current file and included files] (default: ON)

Candidates are selected from the current and included files.

[Search current project] (default: OFF)

Candidates are selected from the current project.

## [Insertion]

Make settings for display and insertion of candidates.

[Insert single proposals automatically] (default: ON)

Selecting this check box automatically enters the candidate if only one candidate is available.

[Insert common prefixes automatically] (default: ON)

Selecting this check box automatically enters common prefixes.

[Present proposals in alphabetical order] (default: OFF)

Selecting this check box lists candidates for content assist in alphabetical order. Deselecting this check box (turned off) lists the candidates in order of suitability, beginning with the most suitable one as determined by factors such as relative position, scope, and prefix.

[Content Assist parsing timeout (ms)] (default: 3000 ms)

In large projects, displaying the candidates for content assist can take some time. You can set a wait time here to permit assist display to time out after a certain interval.

**[Auto activation:]**

You can enter a symbol (".", "->", or ":::") to have content assist start automatically. Deselect the check box to disable this feature.

[Enable "." as trigger] (default: ON)

[Enable "->" as trigger] (default: ON)

[Enable ":::" as trigger] (default: ON)

[delay (ms)] (default: 500 ms)

Specify a time in ms units until content assist is automatically displayed after the above symbol is input.

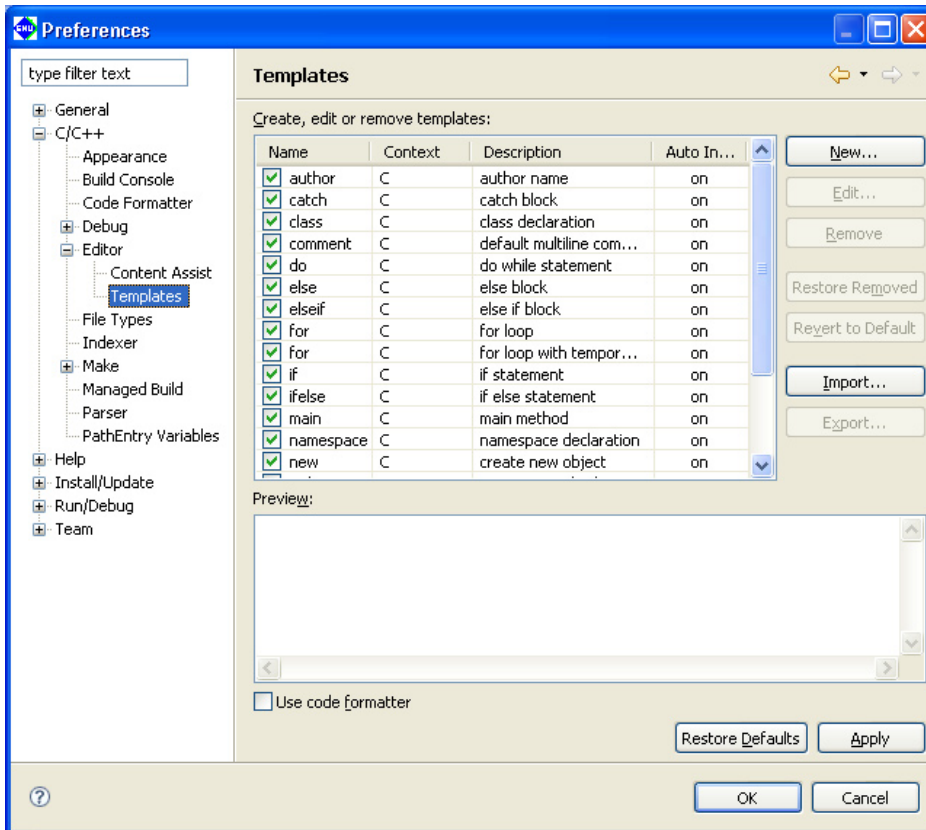
**[Background for completion proposals:]**

Select the background color for the box in which candidates for content assist are listed. Use the button located to the right to select a color.

**[Foreground for completion proposals:]**

Use the button to the right to select the content assist text color.

## C/C++ &gt; Editor &gt; Templates



Manage templates entered with the content assist feature.

[Create, edit or remove templates:]

Lists the defined templates in alphabetical order.

[Name]

This is the template name. The corresponding check box indicates whether the template is enabled or not. The checked templates are enabled and are displayed in the content assist list.

[Context]

Shows a location at which the template can be written. "C" means that the template can be written in C sources.

[Description]

Gives an overview of the template.

[Auto Insert]

"on" means that the template will be automatically inserted if it is only one candidate.

[Preview:]

Displays the contents of the template selected in the list.

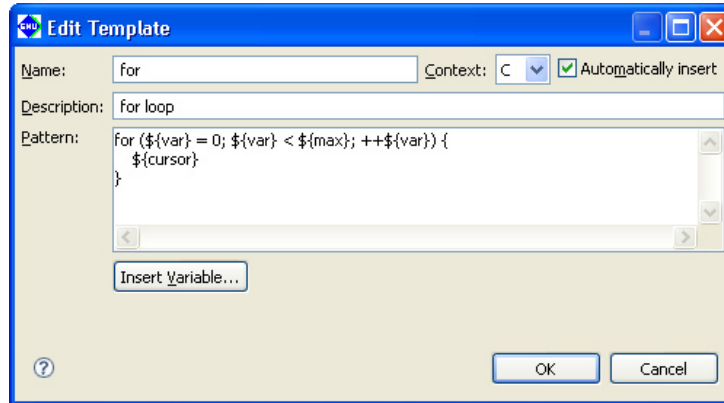
[New...]

Displays the [New Template] dialog box, allowing you to create a new template.

[Edit...]

Displays the [Edit Template] dialog box, allowing you to edit the template selected in the list.

[New/Edit Template] dialog box



Use this dialog box to create or edit a template.

**[Name:]**

Enter a template name.

**[Context:]**

Select context.

**[Automatically insert]**

Select whether the template will be automatically inserted or not.

**[Description:]**

Enter a template description.

**[Pattern:]**

Write a template here.

**[Insert Variable...]**

Shows a list of variables, allowing you to insert a variable indicating a file name, date/time, etc. into the template.

**[OK]/[Cancel]**

Click [OK] to confirm the edited content or [Cancel] to cancel creating or editing a template.

**[Remove]**

Removes the template selected in the list.

**[Restore Removed]**

Restores the removed templates.

**[Revert to Default]**

Reverts the templates to default conditions.

**[Import...]**

Loads a template definition from a file.

**[Export...]**

Writes the contents of the template definition selected in the list to a file. This file can be loaded using the [Import...] button.

## 5.10 Additional Description on Dialog Boxes

---

This section discusses dialog boxes that may require additional description. Not discussed are dialog boxes described in the body text, standard Windows dialog boxes and equivalents, and simple confirmation dialog boxes involving simply [OK] and [Cancel] buttons.

### 5.10.1 Properties for Project

This dialog box is used to display properties and change settings for the currently selected project. This dialog box appears when you select [Properties] from the [Project] menu or from the context menu for the [C/C++ Projects] or [Navigator] view. The 15 categories prepared in the properties list to the left allow you to display the page for a category by clicking from a list. The dialog boxes corresponding to each category page are described below.

Also described below are buttons appearing on every page.

#### [Restore Defaults]

Restores the content of each page to the state in which the dialog box was opened or the state in which the dialog box settings were confirmed with the [Apply] button.

#### [Apply] \*

Applies the changes and content set on the page. Click [Apply] before proceeding to another page if you want to change other properties.

#### [OK] \*

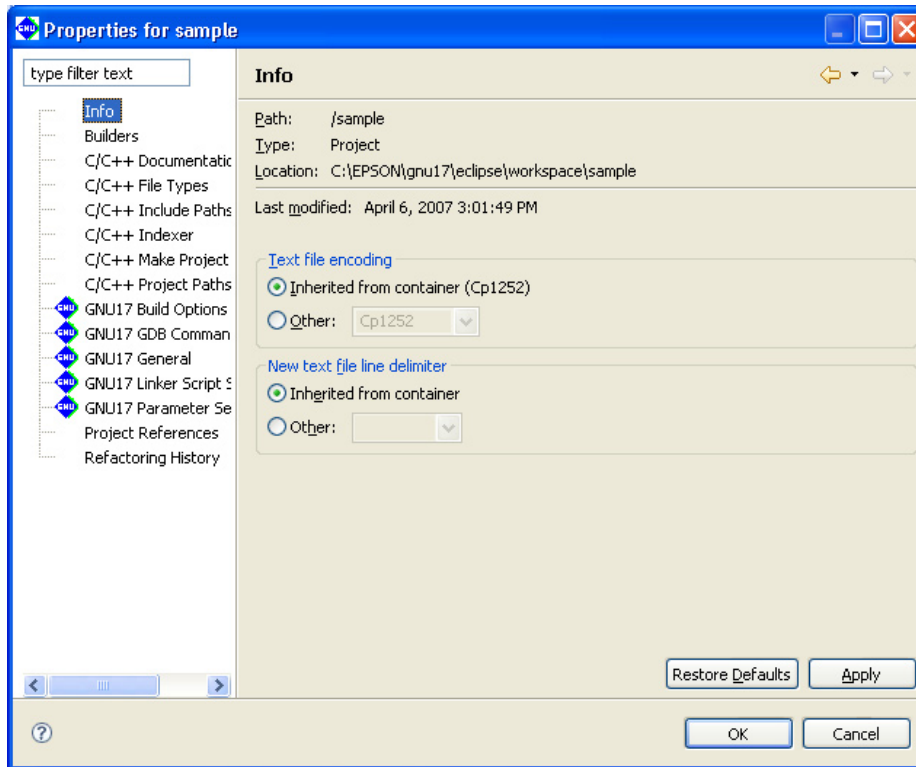
Applies the changes and content set on the current page and closes the dialog box.

#### [Cancel]

Cancels the settings and closes the dialog box. Contents not already confirmed with the [Apply] button before canceling do not revert to their former state.

- \* If you click the [Apply] or [OK] button after settings in a [GNU17 Build Options] or [GNU17 General] page have been changed, a dialog box appears for selecting "clean" build (see Section 5.7.8) to delete the files created with the previous settings (and rebuild).

## Info



Shows the project directory location. You also can set the encoding format and line delimiter for text files such as source files.

## [Text file encoding]

Set text encoding format.

[Inherit from container (\*1)] (\*1: Cp1252, MS932, etc.; varies with Windows language support)

This is the standard Windows character set. (default)

[Other:]

Select another character encoding.

## [New text file line delimiter]

Select a line delimiter. The selection will take effect for files to be created subsequently. It does not affect the existing files.

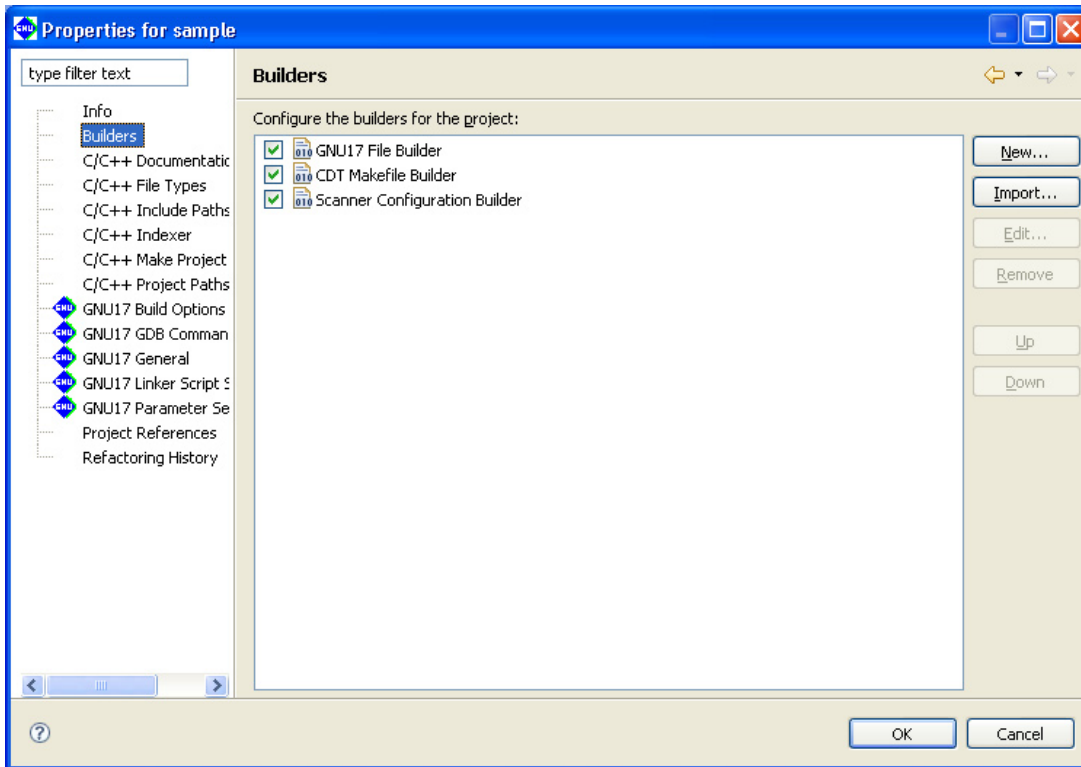
[Inherited from container]

The standard Windows line delimiter is used. (default)

[Other:]

Select another line delimiter.

## Builders



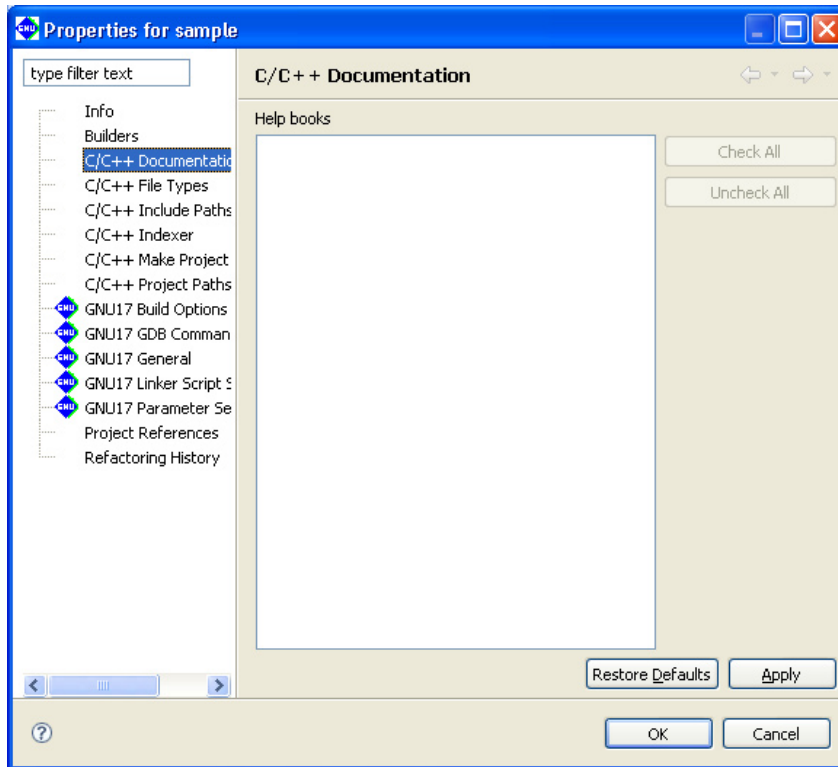
Select the builder for a project.

### [GNU17 File Builder]

Deselect this check box if you do not wish to automatically generate the makefiles and other files needed for a build when executing a build process. (Refer to Section 5.7.9, "Using an Original Makefile".) Otherwise, avoid making any other changes on this page.

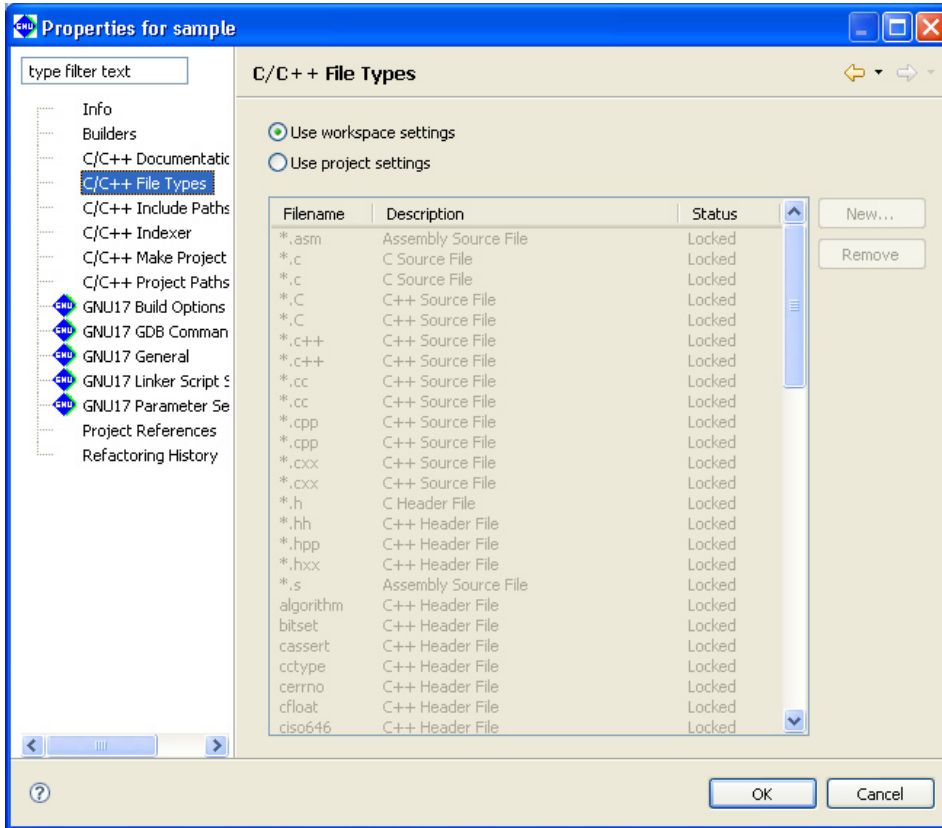


## C/C++ Documentation



Used to select the help documents for the project. No document is used in the **IDE**.

## C/C++ File Types



Define the name or extension of the file you want to be handled as a C resource. There is no need to make changes if you use only files created in the **IDE**.

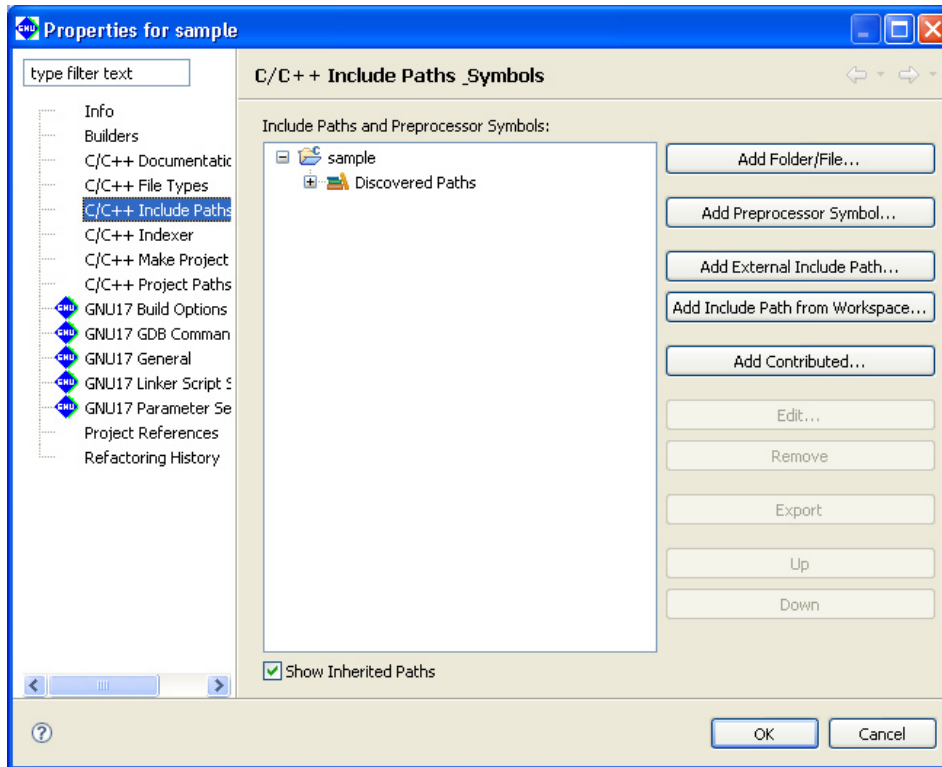
## [Use workspace settings]

Uses settings for file formats shown on the [C/C++] > [File Types] page of the [Preferences] dialog box.

## [Use project settings]

Select this radio button to use a set of file formats specific to a project. Click the [New...] button to add new file extensions. You can remove unnecessary extensions and file names by selecting from the list and clicking the [Remove] button.

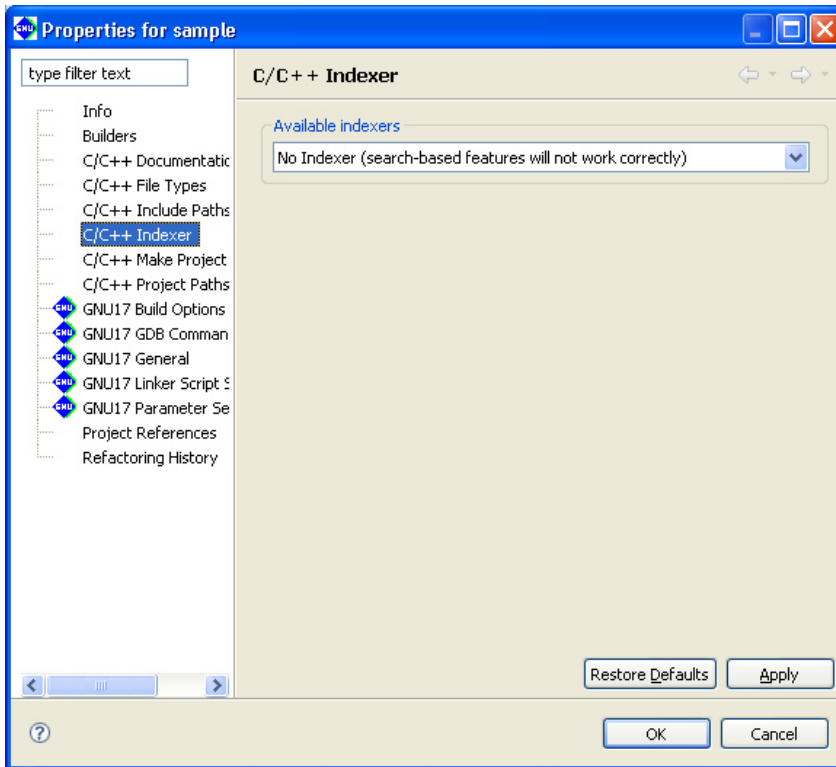
## C/C++ Include Paths & Symbols



Define the path by which to search for an include file or the symbol for a preprocessor.

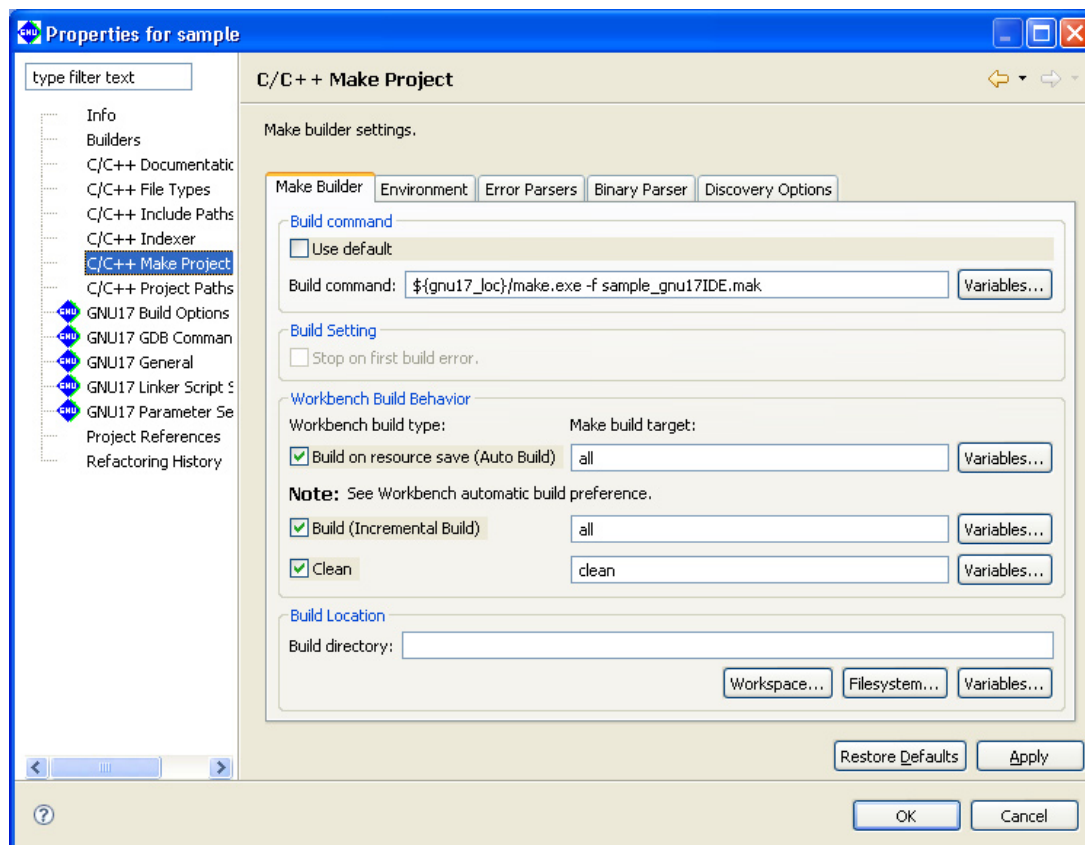
Once a build is executed, the include directory searched during that process and the macro-definitions passed to the preprocessor are added to [Discovered Paths] in the tree list.

**Note:** For normal use, leave this setting unchanged.

**C/C++ Indexer**

Make settings for the indexer used by C search and content assist. Do not change any of the settings on this page.

## C/C++ Make Project



Make settings for the **make.exe** executable that performs the build process. If you are using original makefiles that you created, enter your changes on the page for the [Make Builder] tab. Do not change the pages on other tabs.

## [Build command]

## [Use default]

Leave this check box unselected if you wish to edit the **make.exe** command line.

## [Build command:]

Edit the **make.exe** command line. (Change the makefile to the one you created.)

## [Build Setting]

## [Stop on first build error.]

If this is selected, the **IDE** stops building upon the occurrence of an error. This check box is enabled when [Use default] is selected.

## [Workbench Build Behavior]

## [Build on resource save (Auto Build)]

Specify the target in the makefile to be called during an auto build. By default, "all" is called. (This option is not used with the default **IDE** settings.)

## [Build (Incremental Build)]

Specify the target in the makefile to be called during a build process. By default, "all" is called.

## [Clean]

Specify the target in the makefile to be called during a clean process. By default, "clean" is called.

## [Build Location]

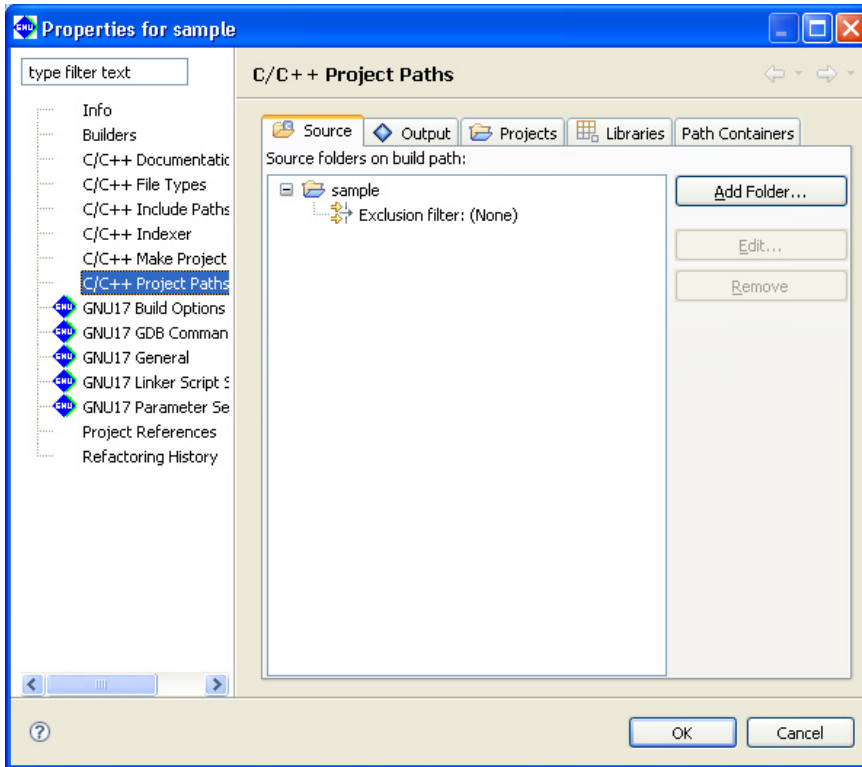
Although this field is used to specify the working directory during a build, no entry is required here.

If you are using your own makefiles, you must also edit or make changes on other pages. For more information, refer to Section 5.7.9, "Using an Original Makefile".

## C/C++ Project Paths

Set the source folder. It is not necessary to set parameters in the pages other than [Source] tab.

### [Source] tab



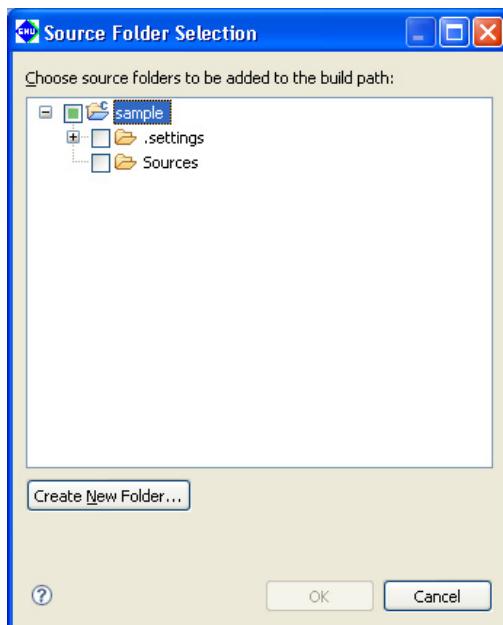
Set the source folder. The source files in the directory set here are written in the makefile used for a build.

### [Source folders on build path:]

This is a list of current source folders. The project directory is listed here by default.

### [Add Folder...]

Adds a source folder to the list.

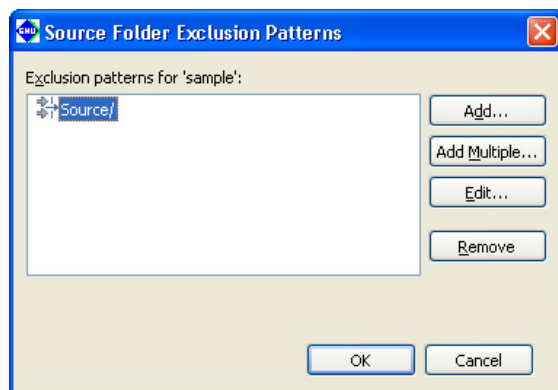


The [Source Folder Selection] dialog box is displayed. Select a source folder from the tree list, or click the [Create New Folder...] button to create a new source folder or link an existing source folder present outside the project directory.

**[Edit...]**

Enter a file name pattern to exclude certain C resources from a source folder to be rendered as source files (files you do not want to include in the makefile).

Select [Exclusion filter:] for the source directory you want to edit from the source directory tree list and click this button. This displays the [Source Folder Exclusion Patterns] dialog box.

**[Exclusion patterns for <source directory>:]**

Lists the file name patterns by which to exclude certain C resources to be rendered as source files (files you do not want to include in the makefile).

**[Add...]**

Adds a file name pattern. You can use "?" as a single-character wildcard and "\*" as a string wildcard.

Example: src?.c The "?" can represent any character. (e.g., src1.c or src2.c)

test.\* The extension can be any string. (e.g., test.c or test.s)

**[Add Multiple...]**

From the list of files in the source directory displayed here, select the files you want to exclude from the source. (Use the [Ctrl] key to make multiple selections.)

**[Edit...]**

Edit the file pattern selected from the list.

**[Remove]**

Removes the file pattern selected from the list.

**[Remove]**

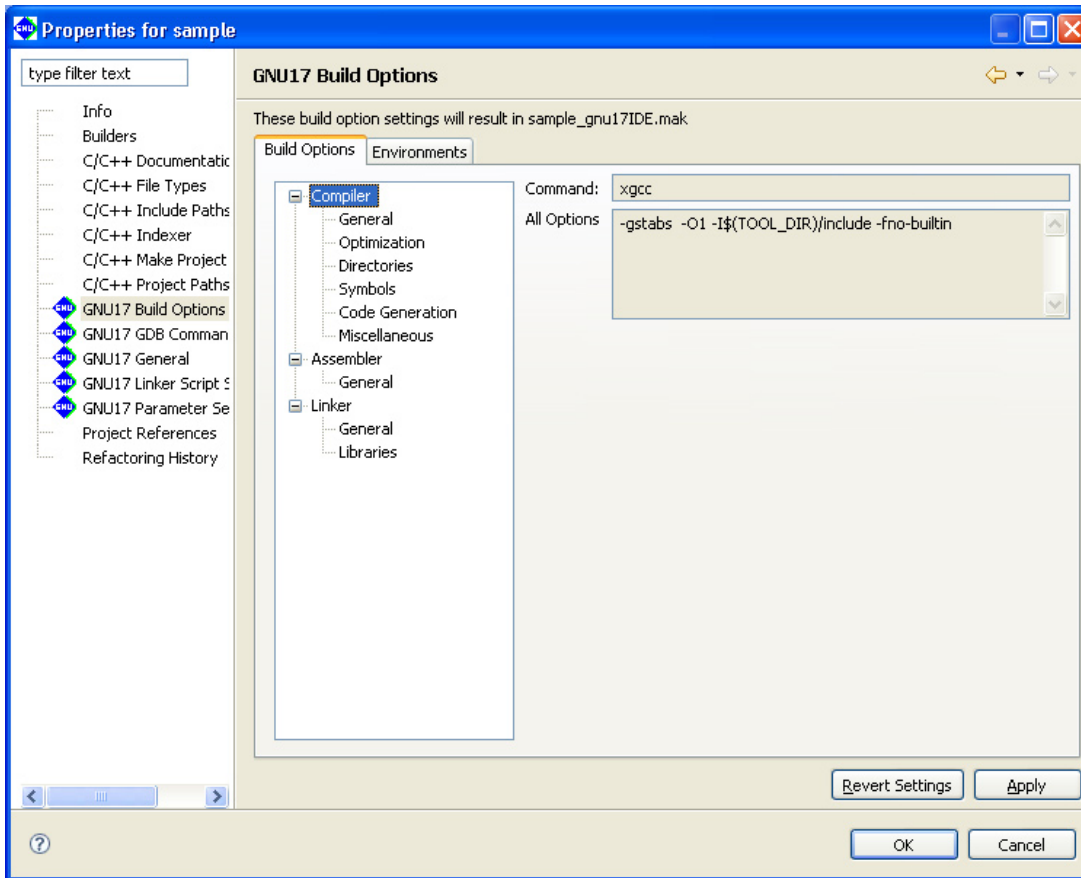
Removes the source directory selected from the list. (The directory is not removed from the file system.)

## GNU17 Build Options

Set command line options for the compiler, assembler, and linker.

- \* If you click the [Apply] or [OK] button after settings in a [GNU17 Build Options] page have been changed, a dialog box appears for selecting "clean" build (see Section 5.7.8) to delete the files created with the previous settings (and rebuild).

### [Build Options] tab > [Compiler]



Shows the current settings for the compiler options.

[Command:]

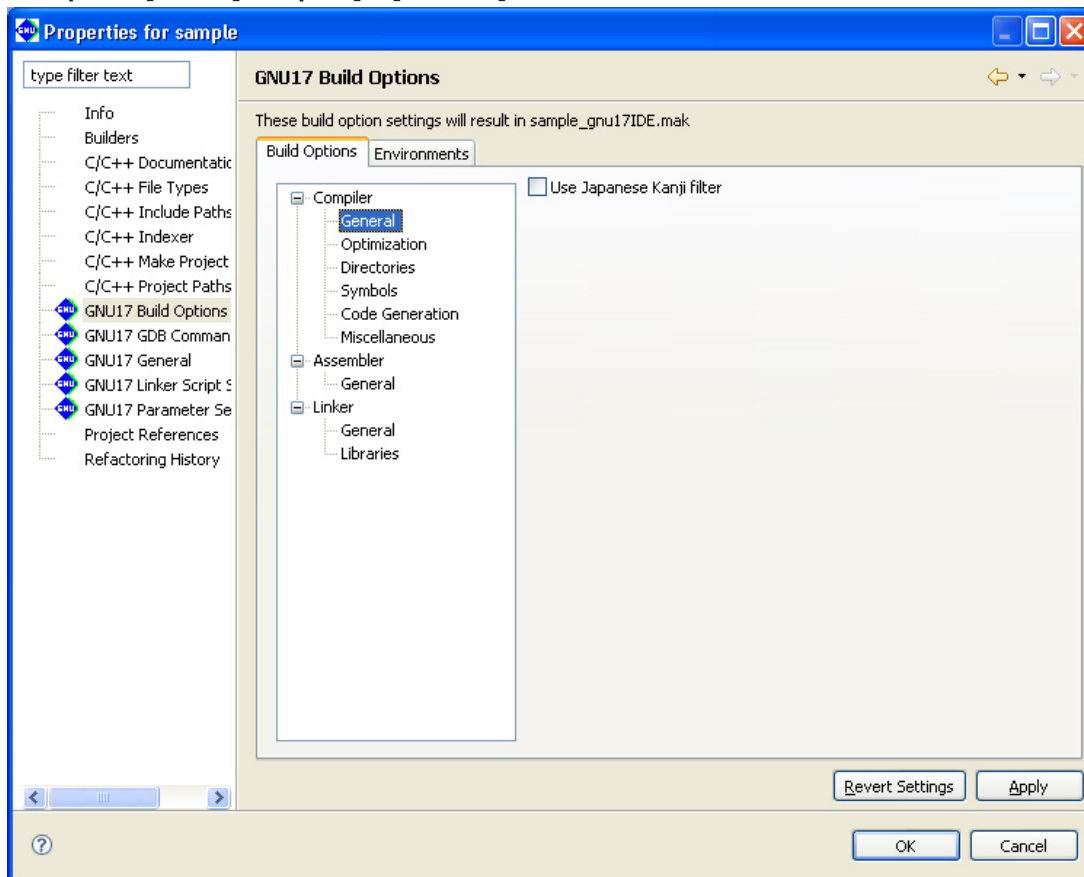
Shows the program name of the C compiler.

[All Options]

Shows the currently set compiler options.



## [Build Options] tab &gt; [Compiler] &gt; [General]



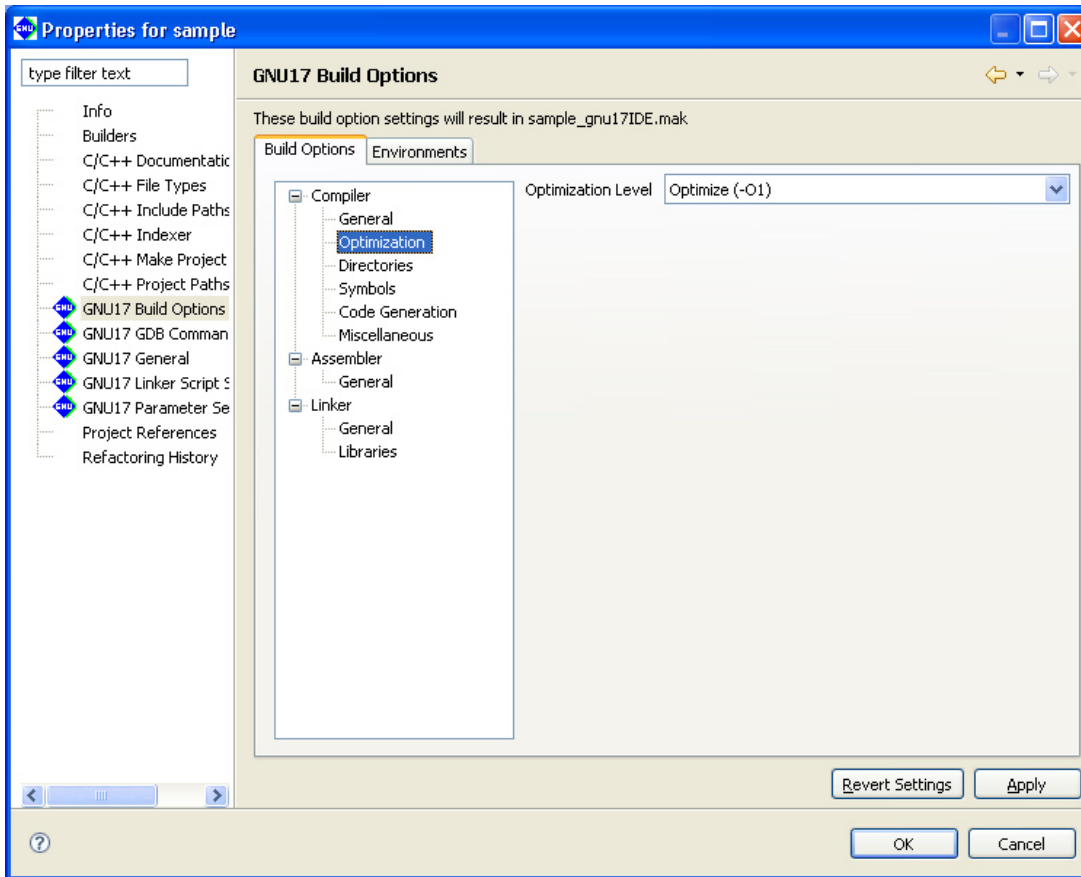
Use this page to select the basic compiler option.

## [Use Japanese Kanji filter]

When this option is specified, the **IDE** converts Shift JIS codes in the source into ASCII escape characters before calling the compiler.

The default check box status depends on the OS under which the **IDE** starts up; the check box is set to on under an OS with Japanese environment or is set to off under other language versions.

## [Build Options] tab &gt; [Compiler] &gt; [Optimization]

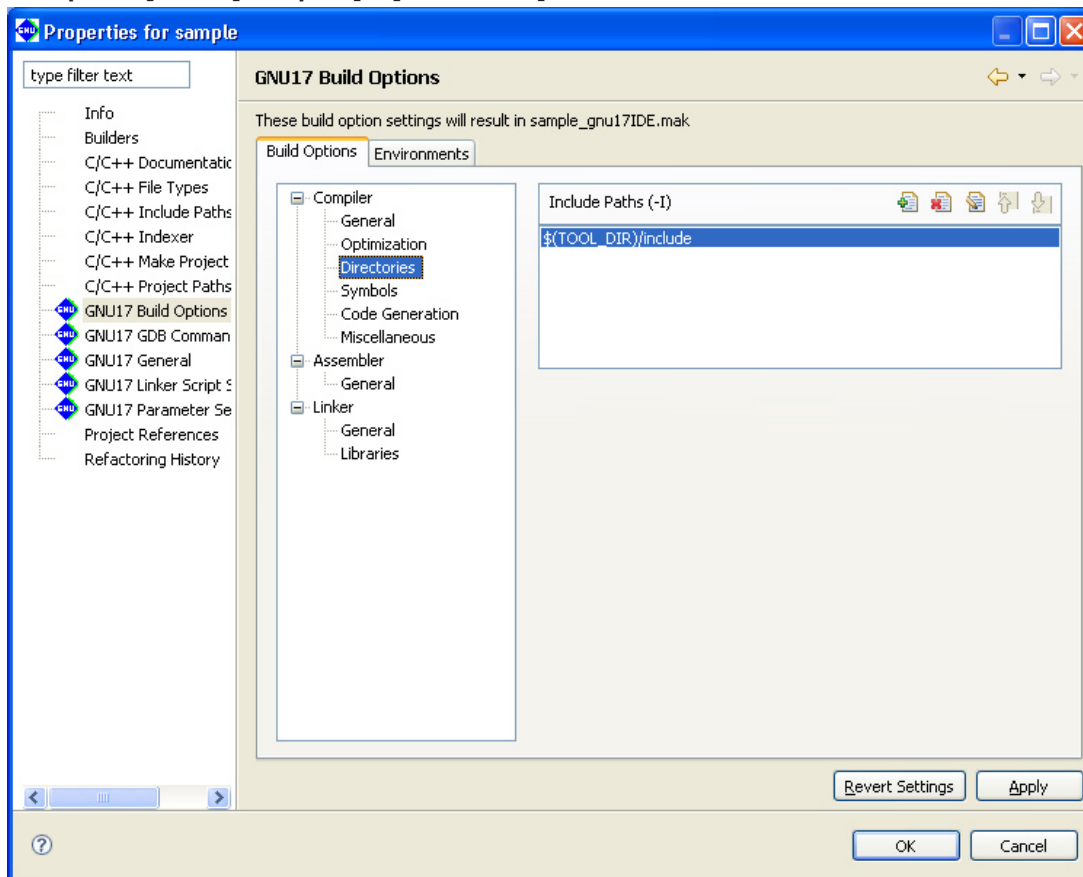


Use this page to select compiler optimization option.

## [Optimization Level]

Select the optimization level (-O1).






## [Build Options] tab &gt; [Compiler] &gt; [Directories]



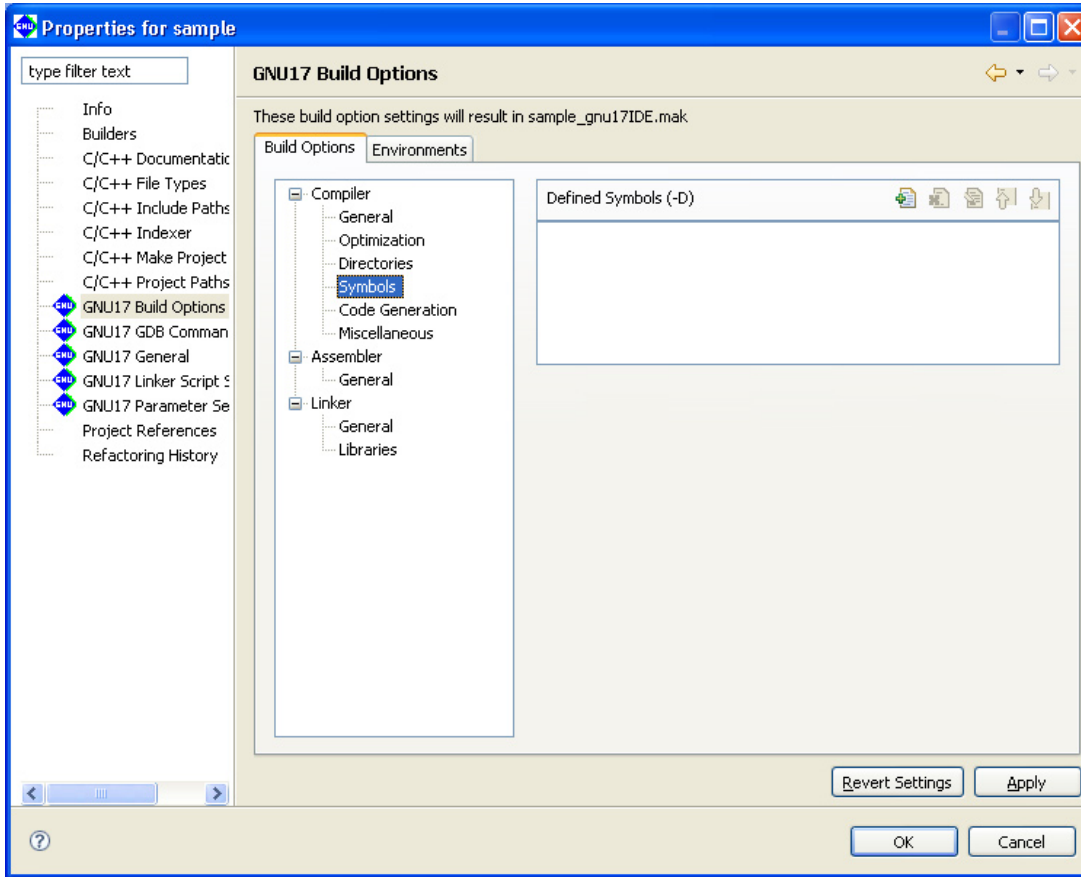
Use this page to set compiler search path options.

## [Include Paths (-I)]

Set the include file search path. The buttons are described below.

-  [Add]      Adds a directory. Displays a dialog box for entering a path or selecting a path with the [Browse...] button.
-  [Delete]      Deletes the path selected in the list.
-  [Edit]      Edits the path selected in the list. Displays a dialog box for editing the path.
-  [Move Up]      Moves the path selected one position up in the list. The include files are searched by order of paths in the list, beginning with the uppermost path.
-  [Move Down]      Moves the path selected one position down in the list.






## [Build Options] tab &gt; [Compiler] &gt; [Symbols]



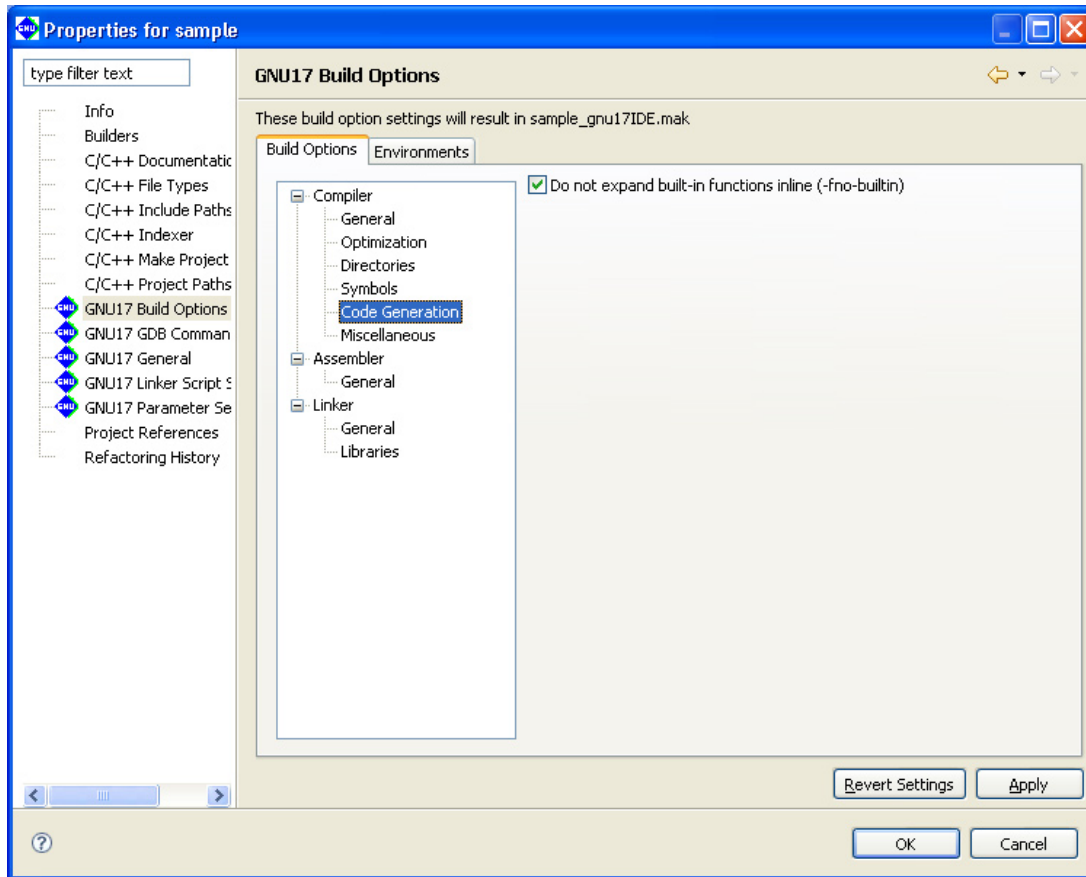
Use this page to set compiler macro-definition options.

## [Defined Symbols (-D)]

Specify a macro-name and replacement string. The buttons are described below.

-  [Add]      Adds a macro-definition. In the dialog box displayed for macro-definition entry, enter a macro-definition in the following format:  
                   <macro-name>  
                   or  
                   <macro-name>=<replacement string>
-  [Delete]      Deletes the macro-definition selected in the list.
-  [Edit]        Edits the macro-definition selected in the list. Displays a dialog box for editing the macro-definition.
-  [Move Up]    Moves the macro-definition selected one position up in the list.
-  [Move Down]   Moves the macro-definition selected one position down in the list.

## [Build Options] tab &gt; [Compiler] &gt; [Code Generation]

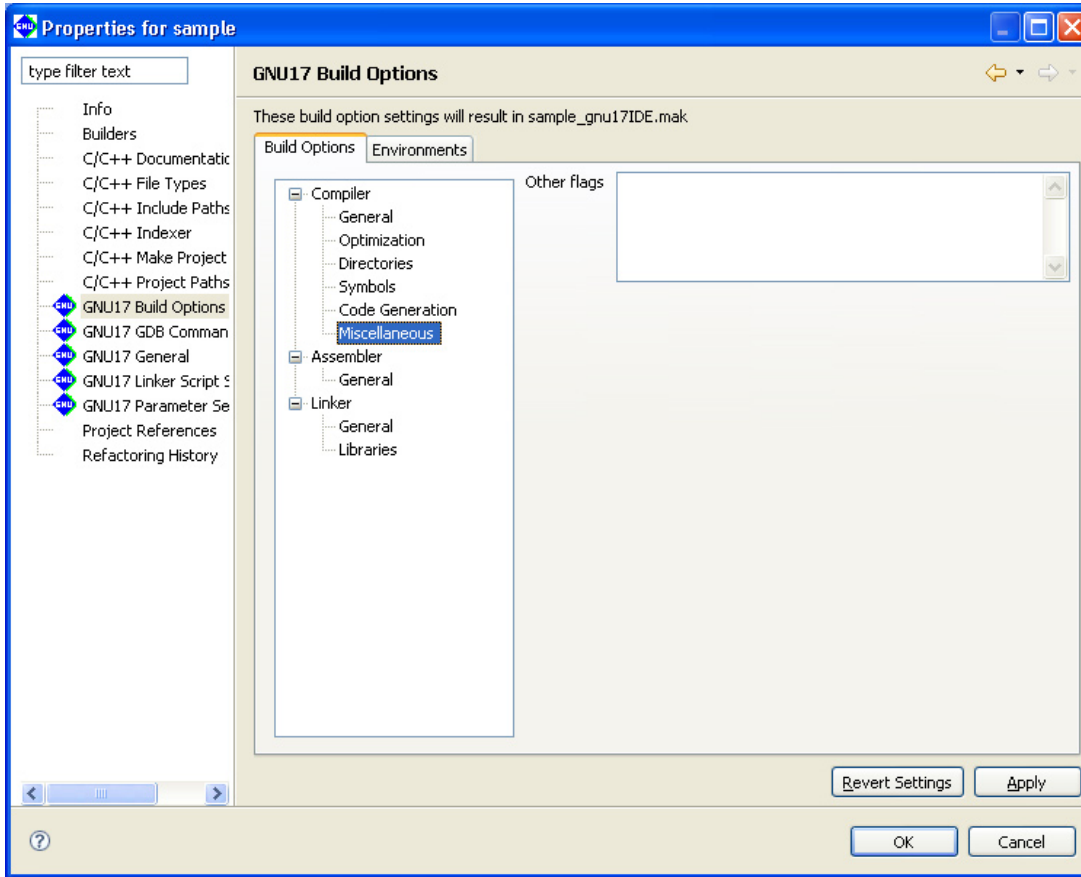


Use this page to select compiler code-generation option.

## [Do not expand built-in functions inline (-fno-builtin)]

When this option is specified, built-in functions are ignored for inline expansion and are always called. For the functions to which this option applies, refer to Section 6.3.2, "Command-line Options".

## [Build Options] tab &gt; [Compiler] &gt; [Miscellaneous]

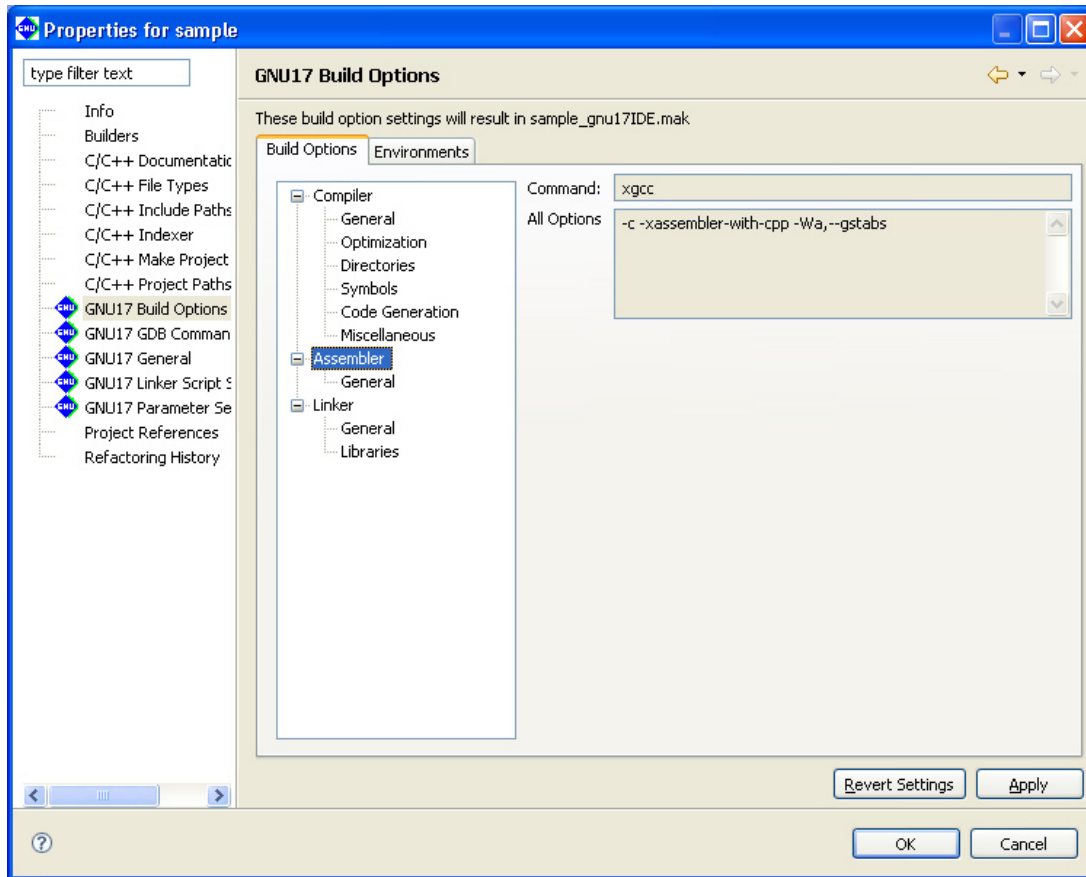


Use this page to set other compiler options.

## [Other flags]

Enter other options directly into this text field. Insert one or more spaces between each option.

## [Build Options] tab &gt; [Assembler]



Shows the current settings for the assembler options.

## [Command:]

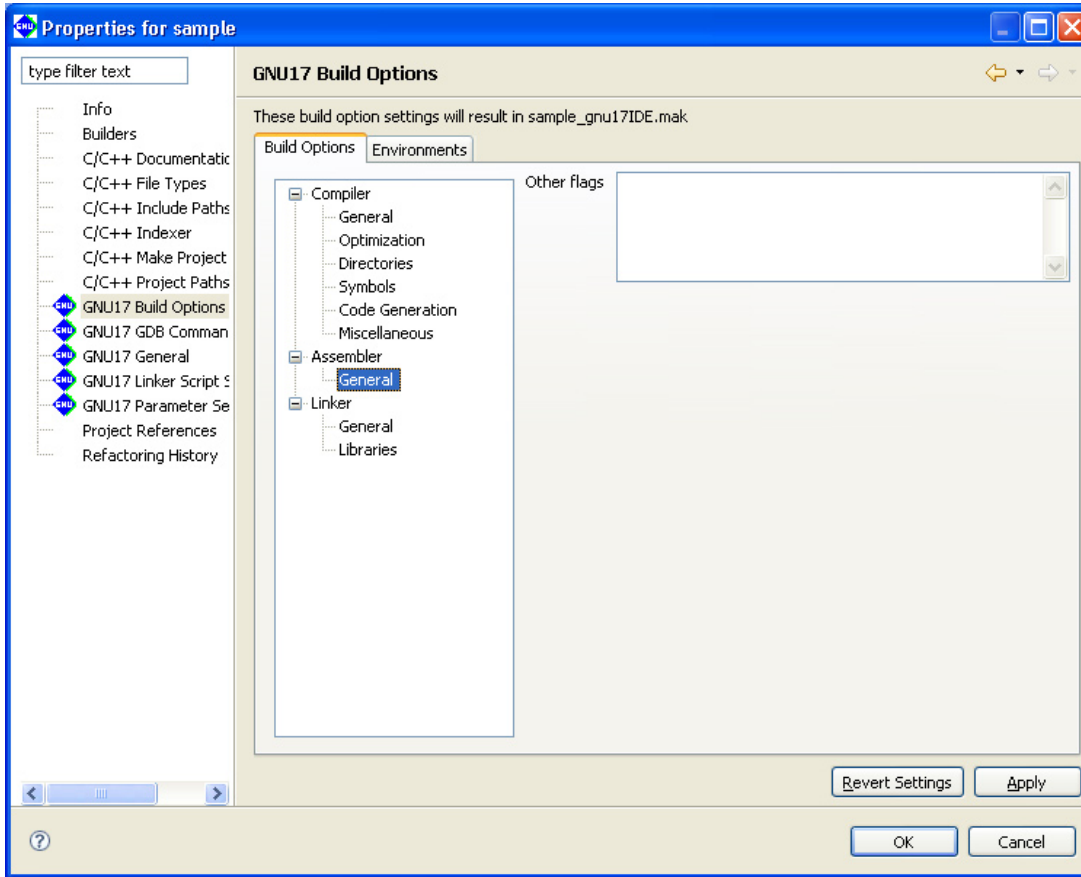
Shows the program name of the compiler\*.

## [All Options]

Shows the currently set options.

\* In the **IDE**, the assembler sources are assembled by the C compiler for which `-xassembler-with-cpp` option is specified.

## [Build Options] tab &gt; [Assembler] &gt; [General]



The `-c`, `-xassembler-with-cpp`, and `-Wa, --gstabs` options (and `-mpointer16` option depending on the memory model selected) are always added. Set other assembler options from this page.

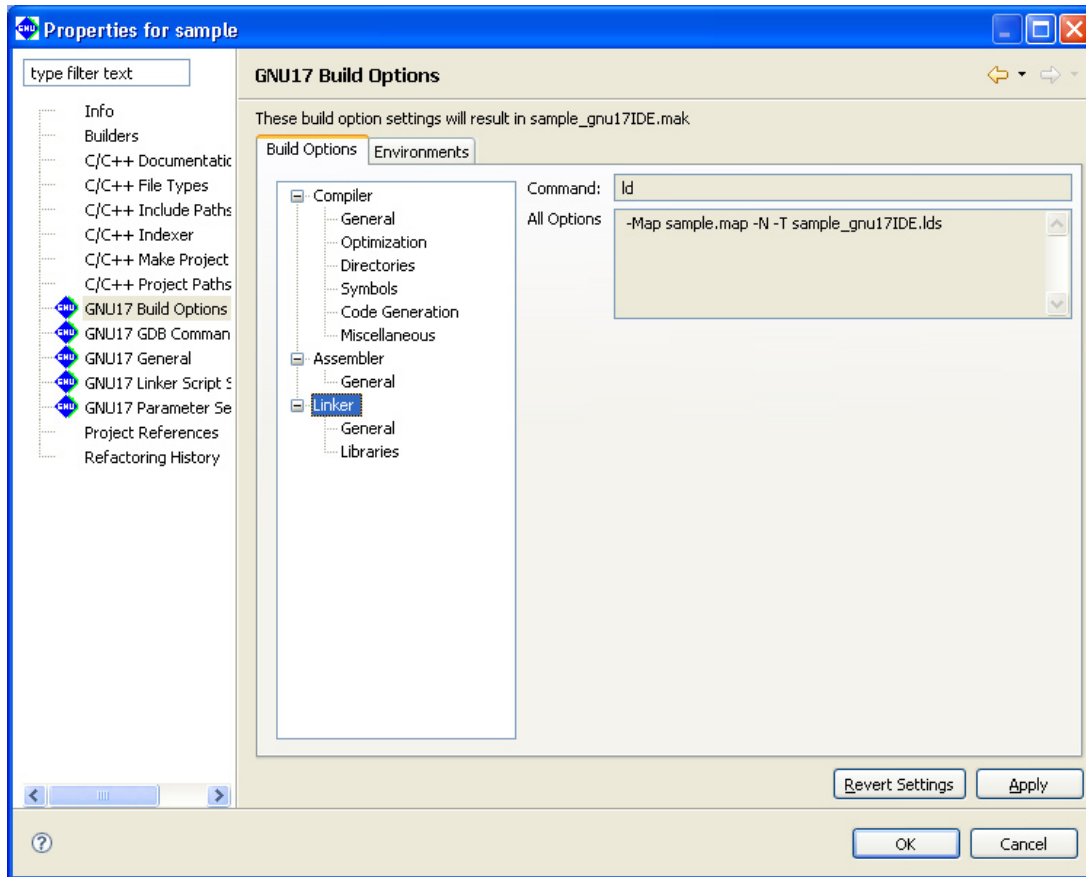
## [Other flags]

Enter the options to be passed to the assembler. Insert one or more spaces between each option.

The options entered are passed to the assembler as "`-Wa, <option>, ...`".



## [Build Options] tab &gt; [Linker]



Shows the current settings for the linker options.

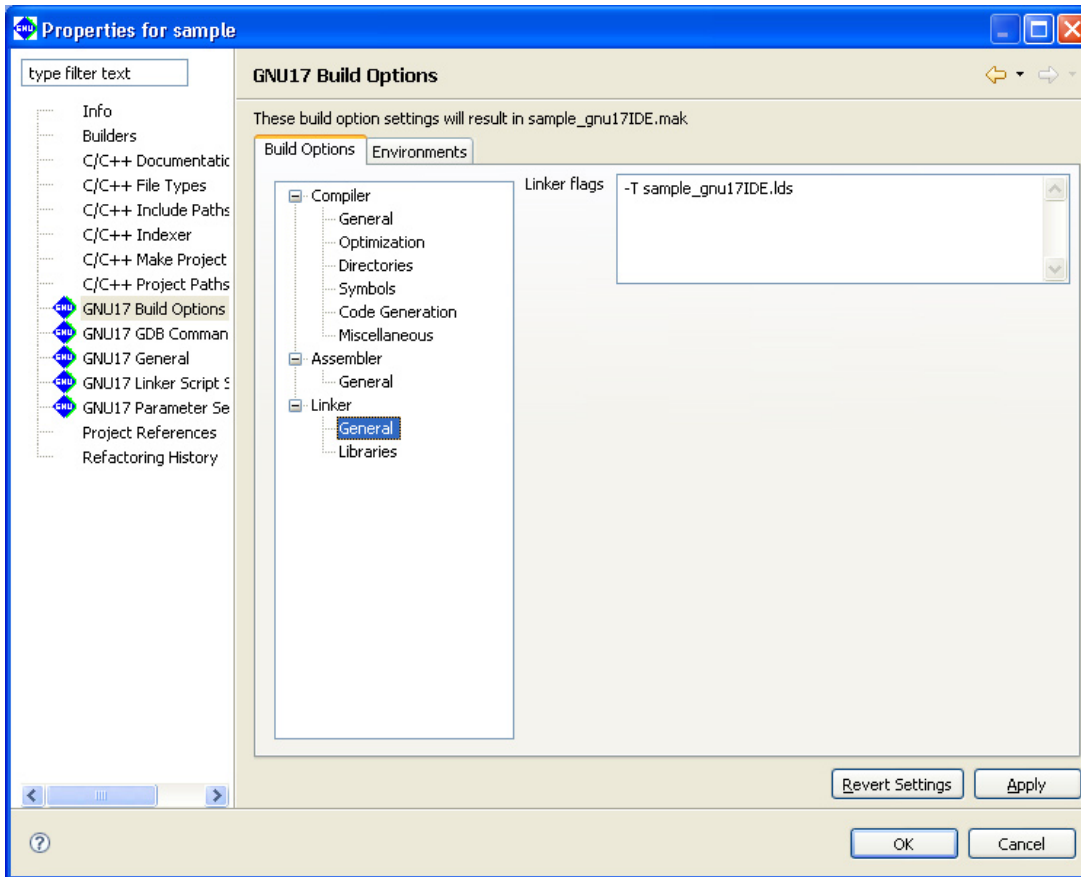
[Command:]

Shows the program name of the linker.

[All Options]

Shows the currently set options.

## [Build Options] tab &gt; [Linker] &gt; [General]

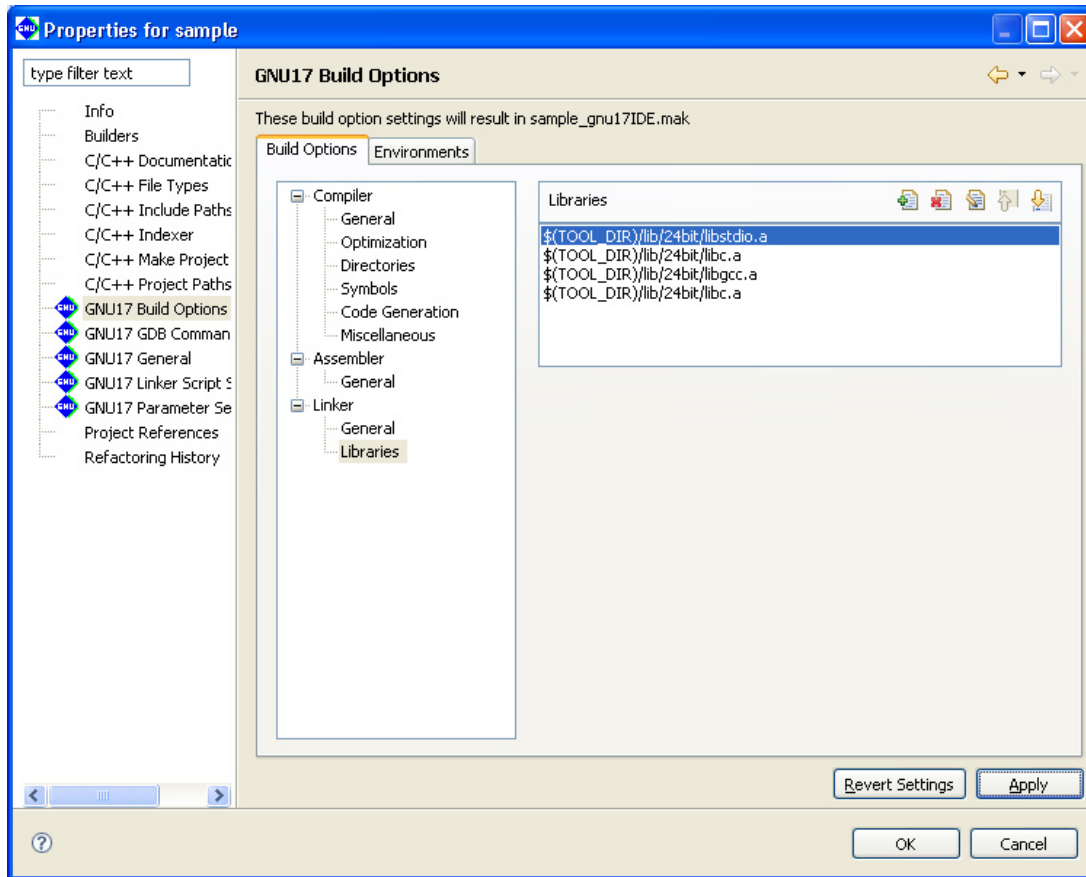


The `-Map` and `-N` options are always added. Set other linker options from this page.

## [Linker flags]

Enter other linker options in this text field. Insert one or more spaces between each option.






## [Build Options] tab &gt; [Linker] &gt; [Libraries]



Use this page to set the libraries to be linked.

[Libraries] (default: `libstdc.a, libc.a, libgcc.a, libc.a*`)

Set the libraries to be linked. The buttons are described below.

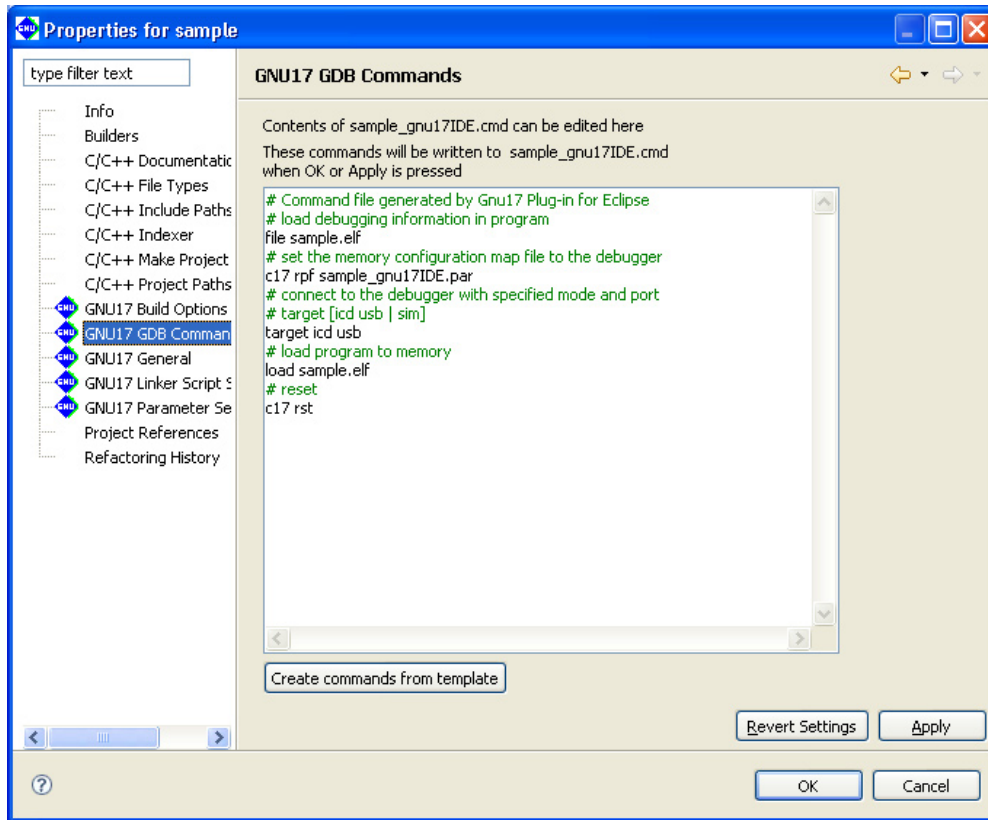
-  [Add] Adds a library. Displays a dialog box for entering a path or selecting one with the [Browse...] button.
-  [Delete] Deletes the selected library in the list.
-  [Edit] Edits the selected library in the list. Displays a dialog box for editing the path.
-  [Move Up] Moves the selected library one position up in the list. Libraries are linked in order of the paths in the list, beginning with the uppermost path.
-  [Move Down] Moves the selected library one position down in the list.

The libraries set here are written in a makefile to link to the objects generated from the sources. However, they must be mapped to sections in a linker script file.

- \* Either the 24-bit libraries or 16-bit libraries are specified by default according to the selected memory model. Furthermore, `libc.a` is specified twice to resolve cross-references between `libc.a` and `libgcc.a`.



## GNU17 GDB Commands

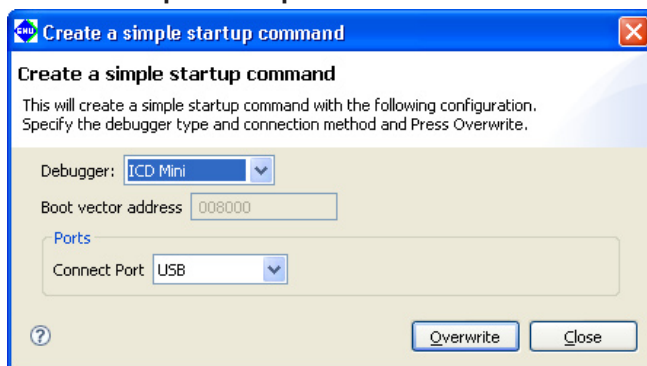


This page is used to edit the debugger start up command file for this project. The contents displayed here are exactly written to the command file.

### [Create commands from template]

Displays the [Create a simple startup command] dialog box shown below to set the connect mode. The command file contents shown in this page reflects the settings in the [Create a simple startup command] dialog box.

### Create a simple startup command



### [Debugger:]

Select the debugger (connect mode) to connect to.  
 ICD Mini When using an ICD to debug.  
 Simulator To debug with a PC only.

### [Boot vector address] (effective only in simulator mode)

Enter the boot vector address in hexadecimal notation (omit 0x) in the text box when creating a command file for simulator mode. The address may be specified in 256-byte increments. The IDE generates a command file for simulator mode that includes a command for setting this value to the boot vector address. The value appeared here by default is the address that was specified when the project was created.

### [Ports]

Selecting ICD Mini or Simulator in [Debugger:] sets the port to USB or NONE, respectively. So no settings are required.

### [Overwrite]

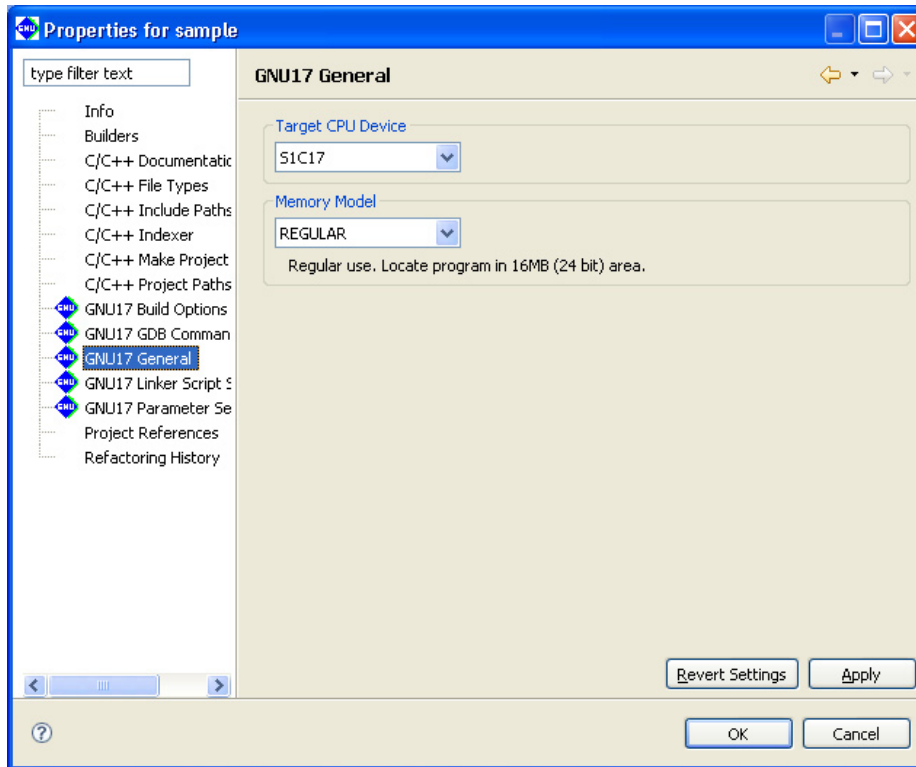
Applies the above settings to the command file shown in the [GNU17 GDB Commands] page and closes the dialog box. A dialog box appears prompting overwrite, so execution may be canceled even after the button is clicked.

### [Close]

Close the dialog box. The command file shown in the [GNU17 GDB Commands] page does not reflect the contents changed in the dialog box.

When you change the target CPU or memory model in the [Properties] > [GNU17 General] page, this dialog box is restored to default settings (for ICD Mini).

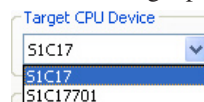
## GNU17 General



Select the target processor and memory model.

## [Target CPU Device]

Select the target processor.



**S1C17:** Built-in S1C17 Core model

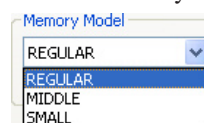
**S1C17701:** When S1C17701 is the target model or when **ES-Sim17** is used for debugging

The models displayed in the list may be added/deleted by the configuration file that will be modified when a new model is released or an existing model is discontinued.

The selected CPU name will be written to the parameter file to be sent to the debugger and used to configure model for debugger's simulator feature. (However, "S1C17" is provided only for core simulation, so it is not described in the parameter file.)

## [Memory Model]

Select a memory model.



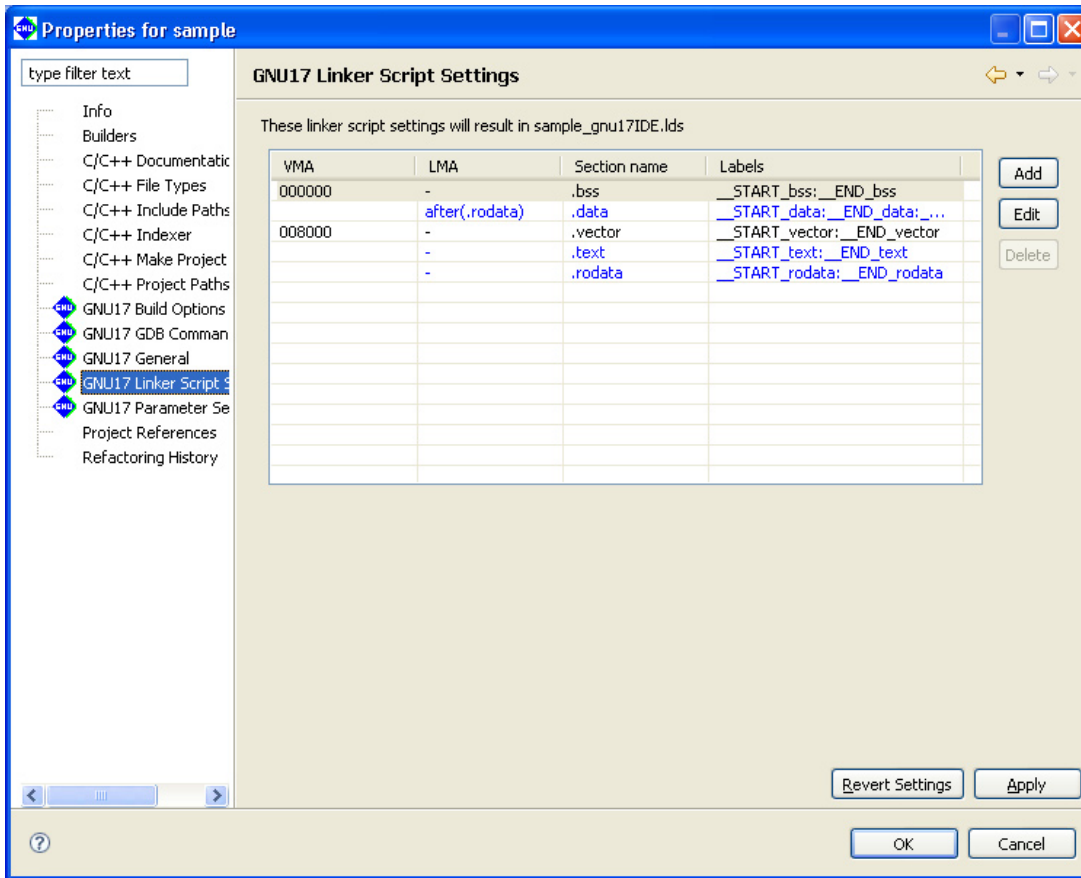
**REGULAR:** 24 bits (Up to 16M-byte space can be used.)

**MIDDLE:** 20 bits (Up to 1M-byte space can be used.)

**SMALL:** 16 bits (Up to 64K-byte space can be used.)

- \* If you click the [Apply] or [OK] button after settings in this page have been changed, a dialog box appears for selecting "clean" build (see Section 5.7.8) to delete the files created with the previous settings (and rebuild).

## GNU17 Linker Script Settings



Edit a linker script.

### Section list

Shows the configuration and location of sections in an execution file (.elf). Information displayed in blue is standard sections defined by default and others displayed in black is user defined sections. To edit the section name, standard section attribute, address to locate, and objects to be located, a user section should be created. The standard section allows the user to specify the location address only, and objects are automatically located except those are located in the user sections with the same attribute.

#### [VMA]

Shows the position (start address) at which a section is placed when it is executed. A section whose address is not written in the VMA will be located at an address following the immediately preceding section.

#### [LMA]

Shows the position in a ROM (start address) at which the actual data is placed. "-" means the same as the VMA (i.e., a section will be executed or accessed from the position at which its actual data is placed). A description "after (<section name>)" means that the actual data for a section will be located following another section indicated in ( ).

#### [Section name]

Indicates the section name.

#### [Labels]

Shows labels indicating the start and the end addresses of the area in which a section will be located. When a LMA is not specified, two labels are displayed. When a LMA is specified, four labels for the start/end VMA addresses and the start/end LMA addresses are displayed, in that order. These labels can be used to specify the address in a source file — for example, when a section is copied from ROM to RAM. The label names are generated automatically from section names.



**[Add]**

Adds section information.

**[Edit]**

Edits the section information selected in the list.

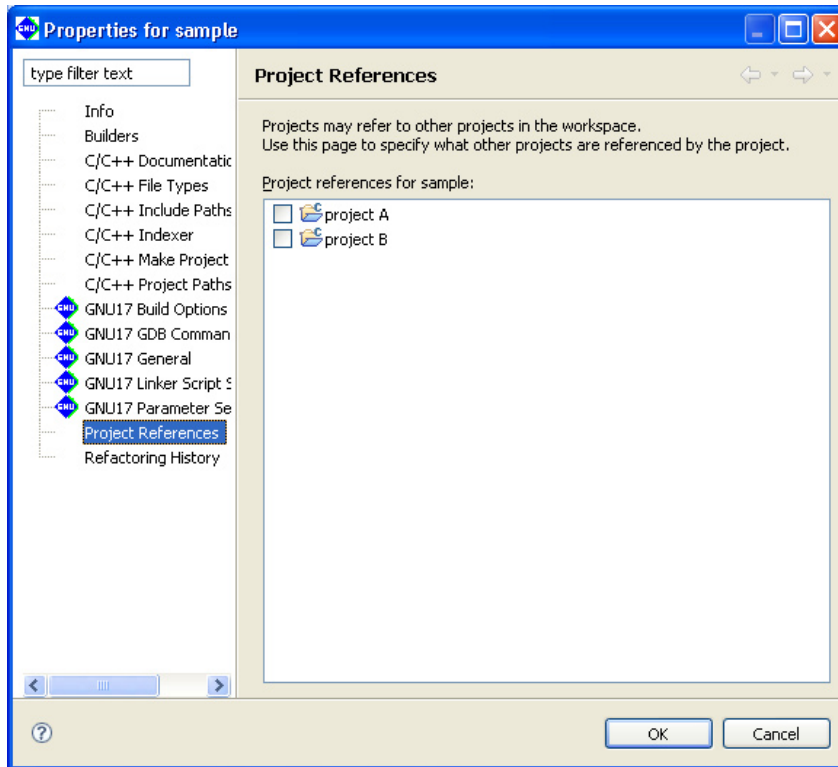
**[Delete]**

Deletes the section information selected in the list.

For more information on how to edit a linker script, refer to Section 5.7.6, "Editing a Linker Script".



## Project References

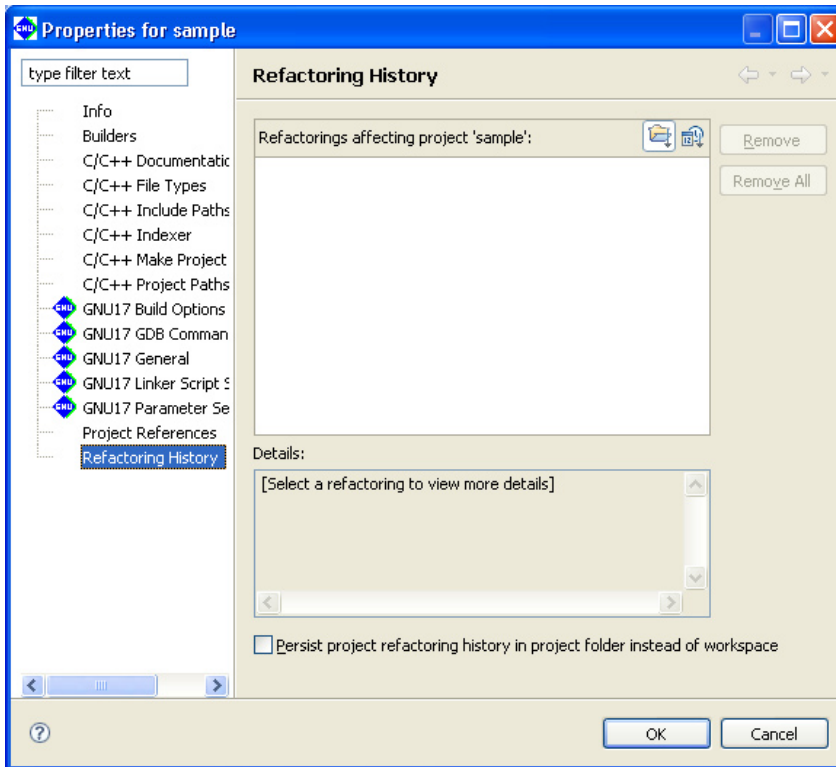


Select the project to be referenced.

[Project references for sample:]

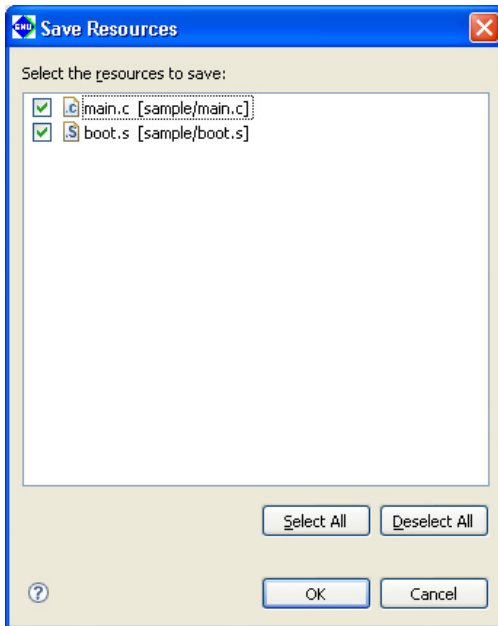
Select the other projects to be referenced by the current project.

## Refactoring History



Shows the refactoring history. The **IDE** does not support this page.

## 5.10.2 Save Resources



This dialog box is displayed if you attempted to close multiple documents before saving the document being edited in the editor.

### File check boxes

Select the check box corresponding to the file you want to save.

### [Select All]

Selects check boxes for all files.

### [Deselect All]

Deselects check boxes for all files.

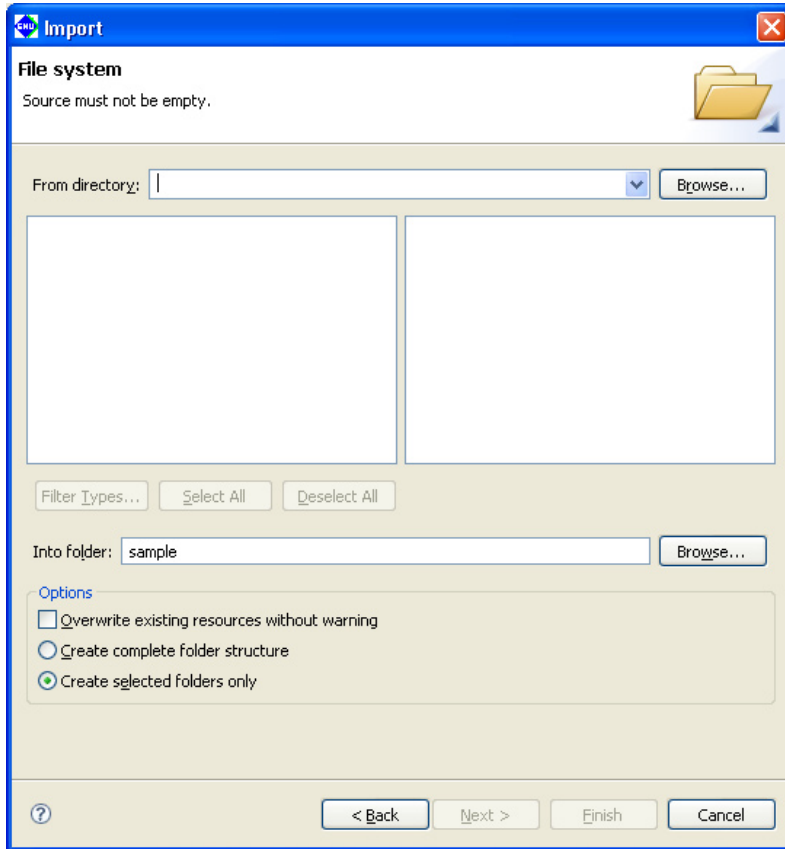
### [OK]

Closes the documents in the editor after saving the selected files.

### [Cancel]

Cancel the action invoking the dialog box. The documents are neither saved nor closed.

### 5.10.3 Import > File system



This dialog box is displayed if you select [File system] in the [Import] dialog box and click the [Next>] button to import a file or directory.

#### [From directory:]

Enter a path to the parent directory for the file or directory to be imported, or select one from the list displayed by clicking the [Browse...] button.

#### Directory list (box to the left)

Lists the subdirectories hierarchically subordinate to the directory selected in [From directory:].

To import a directory, select one from this list.

To import a file, select the parent directory containing it.

#### File list (box to the right)

Lists the files present in the directory selected in the directory list. Use this list to select the file you want to import.

#### [Filter Types...]

Allows you to restrict the import to a subset of the files selected in the file list by specifying a file format (file name extension).

Select a file name extension from the dialog box by clicking this button. Files other than the selected file formats will be deselected.

#### [Select All]

Selects all files displayed in the list box.

#### [Deselect All]

Deselects all files displayed in the list box.

**[Into folder:]**

Shows the project or directory selected in the [C/C++ Projects] or [Navigator] view. To import the selected file or directory into another project or directory, click the [Browse...] button and select from the ensuing dialog box.

**[Overwrite existing resources without warning]**

If this check box is selected, any files or directories at the import destination having the same name are overwritten without warning. If this check box is unselected (default), you will be prompted to confirm that you want to overwrite the files in question.

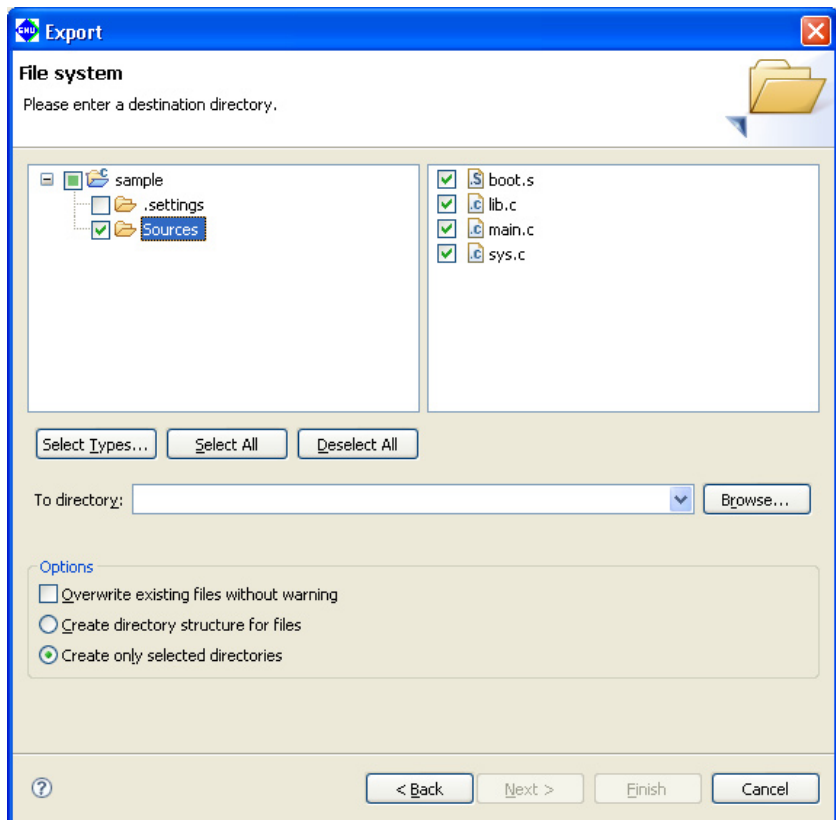
**[Create complete folder structure]**

Selecting this radio button imports the entire directory, including the directory structure of the file system (a tree structure from the root of the selected directory).

**[Create selected folders only]**

If this radio button is selected, only the directory you selected is imported. No directory structures are imported.

## 5.10.4 Export > File system



This dialog box appears if you select [File system] in the [Export] dialog box and click the [Next>] button to export a file or directory.

### Directory list (left-side box)

Lists the subdirectories hierarchically subordinate to the project directory.

To export a directory, select one from this list.

To export a file, select the parent directory that contains it.

### File list (right-side box)

Lists the files currently in the directory selected from the directory list. Use this list to select the file you want to export.

### [Select Types...]

Allows you to export a subset of the files selected in the file list by specifying a file format (file name extension).

Select a file name extension from the dialog box by clicking this button. All files of a different file format are deselected.

### [Select All]

Selects all directories and files displayed in the list box.

### [Deselect All]

Deselects all directories and files displayed in the list box.

### [Overwrite existing files without warning]

If this check box is selected, any files or directories at the export destination having the same name are overwritten without warning. If this check box is unselected (default), you will be prompted to confirm that you want to overwrite the files in question.



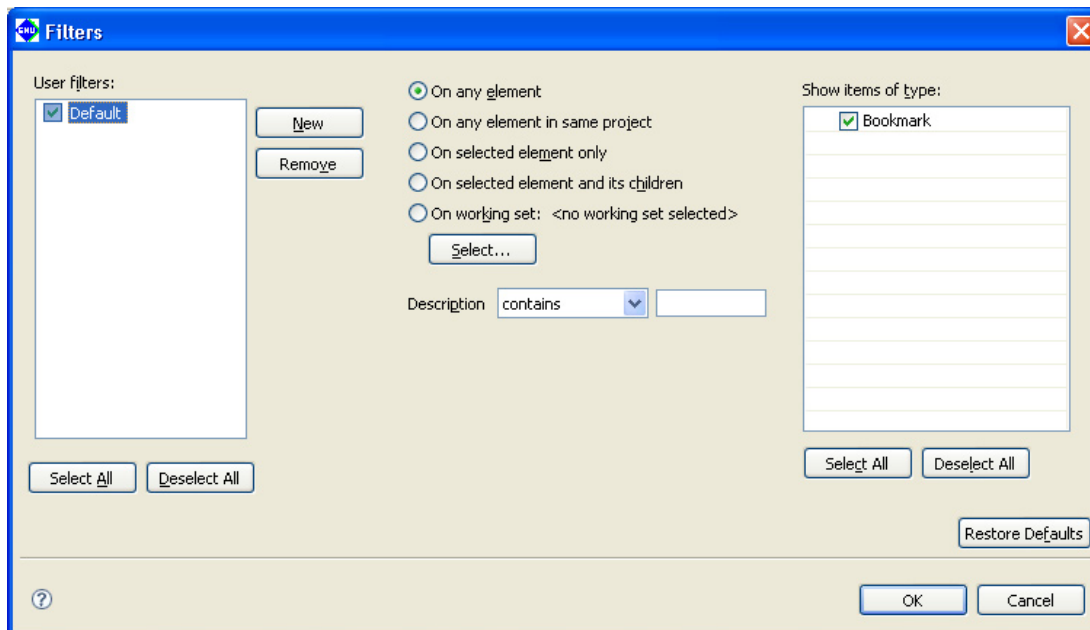
**[Create directory structure for files]**

Selecting this radio button exports the entire file or directory, including the directory structure of the file system (a tree structure from the project directory).

**[Create only selected directories]**

If this radio button is selected, only the directory you selected is exported. The directory structure of the project is not exported.

## 5.10.5 Filters



Click the [Configure Filters] button in [Problems], [Bookmarks], or [Tasks] view to display this dialog box (or select [Configure Filters...] from the view menu). If there are too many items in list view, you can limit the target or number of resources displayed in the list by setting filter parameters here.

### [User filters:]

Displays the filters that have been configured. Use the check boxes to select whether the filter is applied or not. The filters used can also be selected from [Filters] in the view menu.

To modify filtering conditions, select the filter name in this list and select the conditions.

By default, [Default] that displays all elements (its conditions can be modified) is selected.

When two or more filters are selected, all the filtering conditions are ORed. Therefore, when filters are selected with [Default], the filters do not take effect.

### [Select All] (for [User filters:])

Selects all filters displayed in [User filters:].

### [Deselect All] (for [User filters:])

Deselects all filters displayed in [User filters:].

### [New]

Creates a new filter. Enter the filter name in the [Add New Filter] dialog box displayed to add it to the [User filters:] list. Select the filter in the list and set conditions.

### [Remove]

Removes the filters selected in the [User filters:] list.

### [On any element]

All resources in the opened projects are displayed.

### [On any element in same project]

Only resources in the same project as the resources currently active in the editor or the resources selected in the [C/C++ Projects] or [Navigator] view are displayed.

### [On selected element only]

Only resources currently active in the editor or resources selected in the [C/C++ Projects] or [Navigator] view are displayed.

### [On selected element and its children]

Only resources in the directory selected in the [C/C++ Projects] or [Navigator] view are displayed.

**[On working set:]**

Only resources in a specified working set are displayed.

**[Select...]**

Selects the working set displayed if you selected [On working set:].

**[Show items of type:]**

Select the list type to be displayed in the view.

**[Select All]** (for [Show items of type:])

Selects all items displayed in [Show items of type:].

**[Deselect All]** (for [Show items of type:])

Deselects all items displayed in [Show items of type:].

**[Description]**

**contains** Only items containing the string entered in the text box are displayed in [Description].

**doesn't contain** Only items which do not contain the string entered in the text box are displayed in [Description].

Simply leave this text box blank to ignore this restriction.

**[Where severity is:]** ([Problems] view only)

Select if you want to filter the items displayed by severity of error, then select the check box (Error, Warning, or Info) to specify the severity of the items to be displayed.

**[Where priority is:]** ([Tasks] view only)

Select if you want to filter the items displayed by task priority, then select the check box (High, Normal, or Low) to specify the priority of the items to be displayed.

**[Where status is:]** ([Tasks] view only)

To display either completed or incomplete tasks, select this filter option along with the [Completed] or the [Not Completed] radio button.

**[Restore Defaults]**

Returns the above option selections to the default settings.

**[OK]**

Begins filtering with the set conditions.

**[Cancel]**

Cancels filter settings.

## 5.10.6 Sorting



This dialog box is displayed if you select [Sorting...] from the menu for [Problems], [Bookmarks], or [Tasks] view. The list in the view is sorted by the specified condition.

### [Sort by:]

Select the item in the list you wish to prioritize over other items when sorting the list. Four values (1–4) are possible for priority, with value 1 indicating the highest priority. Use the radio buttons to select Ascending or Descending for the order in which you want to sort the selected items.

### [Restore Defaults]

Returns the above conditions to their default settings.

### [OK]

Begins sorting the list with the set conditions.

### [Cancel]

Cancels settings.

## 5.11 Files Generated in a Project by the IDE

Table 5.11.1 List of Files Generated by the IDE

File name	File type	Editing	File management
.project	IDE project file	Prohibited	Required
.cdtproject	IDE project file (CDT)	Prohibited	Required
.gnu17project	IDE project file (GNU17)	Prohibited	Required
GDB17 Launch for <project name>.launch	GDB launch setting file	Prohibited	Required
<project name>_gnu17IDE.cmd	GDB command file	Possible *1	Required
\.settings	Project settings directory	Prohibited	Required
\.externalToolBuilders	Project settings directory (builder)	Prohibited	Required
<project name>_gnu17IDE.mak	makefile	Prohibited	Not required
<project name>_gnu17IDE.lids	Linker script file	Prohibited	Not required
<project name>_gnu17IDE.par	Parameter file	Prohibited	Not required
<source file name>.elf	Executable file	Prohibited	Not required
<source file name>.map	Map file	Prohibited	Not required
<source file name>.dump	Symbol file for two-pass make	Prohibited	Not required
<source file name>.d	Dependency file for makefile	Prohibited	Not required
<source file name>.o	Object file	Prohibited	Not required
<source file name>.ext0	Assembler source file for two-pass make	Prohibited	Not required

\*1: Can be edited using an editor only when the [Properties] dialog box for the project is closed.

The files with "Required" in the "File management" column must be managed using a source management application.

THIS PAGE IS BLANK.

# 6 C Compiler

This chapter explains how to use the **xgcc** C compiler, and provides details on interfacing with the assembly source. For information about the standard functions of the C compiler and the syntax of the C source programs, refer to the ANSI C literature generally available on the market.

## 6.1 Functions

The **xgcc** C compiler compiles C source files to generate an assembly source file that includes S1C17 Core instruction set mnemonics, extended instructions, and assembler directives. The **xgcc** is a gnu C compiler in conformity with an ANSI standard. Since special syntax is not supported, the programs developed for other types of microcomputers can be transplanted easily to the S1C17 Family.

Furthermore, since this C compiler has a powerful optimizing capability that allows it to generate a very compact code, it is best suited to developing embedded applications.

This C compiler consists of three files: **xgcc.exe**, **cpp.exe** and **cc1.exe**.

The **xgcc** is based on the C compiler of Free Software Foundation, Inc. Details about the license of this compiler are written in the text file "Copying GNU", therefore, be sure to read this file before using the compiler.

## 6.2 Input/Output Files

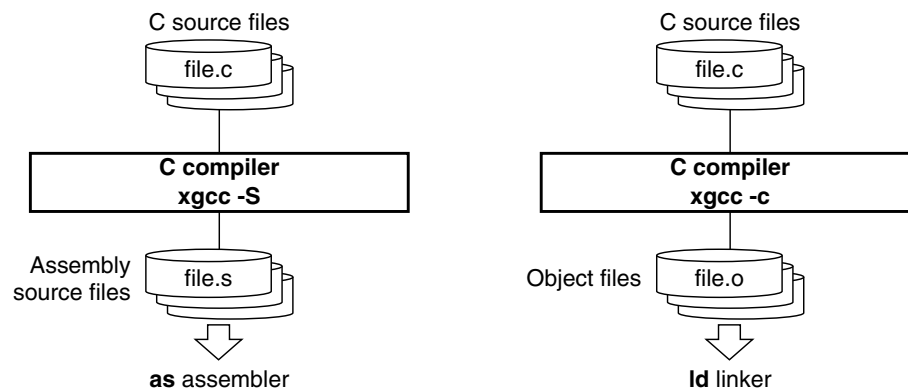


Figure 6.2.1 Flowchart

### 6.2.1 Input File

#### C source file

File format: Text file

File name: `<file name>.c`

Description: File in which the C source program is described.

### 6.2.2 Output Files

#### Assembly source file

File format: Text file

File name: `<file name>.s`

Description: An assembly source file to be input to the **as** assembler. This file is generated when the `-S` option is specified.

#### Object file

File format: Binary file

File name: `<file name>.o`

Description: A relocatable object file to be input to the **ld** linker. This file is generated when the `-c` option is specified.

**Note:** The **xgcc** C compiler generates an elf format executable object file or preprocessed source file according to the option specified.

## 6.3 Starting Method

---

### 6.3.1 Startup Format

To invoke the **xgcc** C compiler, use the command shown below.

```
xgcc <options> <file name>
```

<options> See Section 6.3.2.

<file name> Specify C source file name(s) including the extension (.c).

### 6.3.2 Command-line Options

The compiler provided in this package formally supports the command line options described below.

All other command line options lie beyond the scope of the performance guarantee, and use thereof is solely the user's responsibility.

**-c**

Function: **Output relocatable object file**

Description: This option is used to output a relocatable object file (<input file name>.o). When this option is specified, the **xgcc** C compiler stops processing after the stage of assembly has finished and does not link. The basic make file generated by the **IDE** specifies this option when invoking the **xgcc** C compiler. Do not specify the **-S** or **-E** option simultaneously when this option is used.

Default: The **xgcc** C compiler generates the elf executable object file.

**-S**

Function: **Output assembly code**

Description: This option is used to output an assembly source file (<input file name>.s). When this option is specified, the **xgcc** C compiler stops processing after the stage of compilation has finished and does not assemble the compiled code. Do not specify the **-c** or **-E** option simultaneously when this option is used.

Default: The **xgcc** C compiler generates the elf executable object file.

**-E**

Function: **Execute C preprocessor only**

Description: When this option is specified, the **xgcc** C compiler stops processing after the stage of preprocessing has finished and does not compile or assemble the preprocessed code. The results are output to the standard output device. Do not specify the **-S** or **-c** option simultaneously when this option is used.

Default: The **xgcc** C compiler generates the elf executable object file.

**-B<directory>**

Function: **Specify compiler search path**

Description: This option is used to add <directory> to the search paths of the **xgcc** C compiler.

Input <directory> immediately after **-B**. Multiple directories can be specified. In this case, input as many instances of **-B<directory>** as necessary. The sub-programs (**cpp**, **cc1**, etc.) and other data files of the compiler itself are searched in the order they appear in the command line.

File search is performed in order of priorities, i.e., current directory, **-B** option, and **PATH** in that order.

Default: The **xgcc** C compiler searches sub-programs in the current directory and the **PATH** directory.



**-I<directory>**

Function: **Specify include file directory**

Description: This option is used to specify the directory that contains the files included in the C source.

Input <directory> immediately after -I. Multiple directories can be specified. In this case, input as many instances of -I<directory> as necessary. The include files are searched in the order they appear in the command line.

If the directory is registered in environment variable C\_INCLUDE\_PATH, the -I option is unnecessary.

File search is performed in order of priorities, i.e., current directory, -I option, and C\_INCLUDE\_PATH in that order.

Default: The **xgcc** C compiler searches include files in the current directory and the C\_INCLUDE\_PATH directory.

**-D<macro name> [=<replacement character>]**

Function: **Define macro name**

Description: This option functions in the same way as #define. If there is =<replacement character> specified, define its value in the macro. If not specified, the value of the macro is set to 1.

Input <macro name>[=<replacement character>] immediately after -D. Multiple macro names can be specified. In this case, input as many instances of -D<macro name>[=<replacement character>] as necessary.

**\* About automatic generation of macro names**

The macro names listed below are automatically defined during compilation. These macro names can be referenced from any source file. Note, however, that the same macro names cannot be used for macro definitions in the user program.

Macro name	Contents
__c17	Indicates that the source was compiled for S1C17 processors.
__INT__	Indicates the data size of int type variables (16).
__POINTER24	Indicates that the source was compiled without the -mpointer16 compile option specified.
__POINTER16	Indicates that the source was compiled with the -mpointer16 compile option specified.
__LONG_OFFSET	Indicates that the source was compiled without the -mshort-offset compile option specified.
__SHORT_OFFSET	Indicates that the source was compiled with the -mshort-offset compile option specified.

**-O**

Function: **Optimization**

Description: Optimizes code generation by prioritizing speed and size.

The optimization performed here includes the following processes:

Unnecessary code is deleted (such as code that assigns a value to a local variable that is never referenced).

Variable processing is assigned a register, and the value of this register is reused to reduce memory read/write counts.

However, since this removes the guarantee for memory accesses, variables that require fail-proof read/write to memory must be declared with `volatile`.

Code generation is optimized by predicting branch conditions, preventing repetition of duplicate compare instructions.

Do not use the -O0, -O2, -O3, and -Os options, as they are not supported in this compiler.

Default: Code optimization is performed.

**-gstabs**

Function: **Add debugging information with relative path to source files**

Description: This option is used to create an output file containing debugging information.

The source file location information is output as a relative path. The basic make file generated by the IDE specifies this option when invoking the **xgcc** C compiler.

Default: No debugging information is output.

**-fno-builtin**

Function: **Disable built-in functions**

Description: The functions listed below are always called, not compiled as built-in functions. If this option is not specified, the compiler will expand the following functions inline or replace them with other functions make code generation more efficient, depending on circumstances.

abort, abs, cos, exit, exp, fabs, fprintf, fputs, labs, log, memcmp, memcpy, memset, printf, putchar, puts, scanf, sin, sprintf, sqrt, sscanf, strcat, strchr, strcmp, strcpy, strcspn, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strstr, vprintf, vsprintf

Default: The built-in functions are enabled.

**-mpointer16**

Function: **Generate code for 16-bit (64KB) data space**

Description: This option is used to generate codes that use 16-bit data pointers (the data space is limited up to 64KB).

This option allows the user program to reduce the RAM size for storing static variable pointers. However, the stack size cannot be reduced by this option.

Default: The C compiler generates the object that allows data to be located in the 24-bit (16MB) space.

**-mshort-offset**

Function: **Generate code for 20-bit (1MB) space**

Description: This option is used to generate codes in which the data space and branch address range are limited to 20-bit (1MB space).

When this option is specified, the C compiler uses only one `ext` instruction for extending the immediate data in the data access/branch instructions to reduce code size. This limits the range of data accessing and conditional branching to a 1MB space. However, the `call` and `jpr` branch instructions are not limited to 1MB, as one `ext` instruction extends them to branch within a 24-bit range. This option is useful to reduce code size for applications of which the program and data can be located in a 1MB space. The C compiler always outputs 's' extended instructions (`sld`, `sjreq`, etc.), not 'x' extended instructions (`xld`, `xjreq`, etc.).

Default: The C compiler generates objects that allow data/program to be located in the 24-bit (16MB) space.

Table 6.3.2.1 -mpointer16 and -mshort-offset option settings

Option	Data/Program space		
	24 bits (16MB) *1 Code is generated with a 24-bit data pointer	20 bits (1MB) *1 Code is generated with a 24-bit data pointer	16 bits (64KB) *2 Code is generated with a 16-bit data pointer
	Use entire space	Reduce code size	Reduce code and RAM size
-mpointer16	Do not specify (default)	Do not specify (default)	Specify *3
-mshort-offset	Do not specify (default)	Specify	Specify *3

\*1 Use the ANSI C and emulation libraries for 24-bit memory model (located in `lib\24bit`).

\*2 Use the ANSI C and emulation libraries for 16-bit memory model (located in `lib\16bit`).

\*3 Use the `-mpointer16` option in conjunction with the `-mshort-offset` option except when data/program can be located in a 16-bit (64KB) space.

**-xassembler-with-cpp**

Function: **Invoking C preprocessor**

Description: When this option is specified, the **cpp** C preprocessor will be executed before the source is assembled. This allows assembly sources to include preprocessor instructions (`#define`, `#include`, etc.).

Default: The C preprocessor is not invoked.

**-Wa, <option>**

Function: **Specify an assembler option**

Description: The specified option will be passed to the assembler. To specify two or more options, input as many instances of `-Wa, <option>` as necessary.

Default: No option will be passed to the assembler.

When entering options in the command line, you need to place one or more spaces before and after the option.

Example: `xgcc -c -gstabs test.c`

- Notes:**
- Be aware that the compile processing will be unsteady if the same compiler option is specified twice or more with different settings.
  - Be sure to specify one of the `-S`, `-E` or `-c` options when invoking **xgcc**. If none are specified, **xgcc** continues processing until the linkage stage. Note, however, that necessary linker options cannot be specified in this case.
  - Generate all the objects to be linked using the same `-mpointer16` and `-mshort-offset` option combination in compiling and assembling. Objects that were generated with different option specifications may not be linked normally.

**<Linkable combination>**

Object 1	Object 2
Any	Same specification as object 1
Default	<code>-mshort-offset</code>
<code>-mpointer16</code>	<code>-mpointer16</code> and <code>-mshort-offset</code>

**<Unlinkable combination>**

Object 1	Object 2
Default	<code>-mpointer16</code>
Default	<code>-mpointer16</code> and <code>-mshort-offset</code>
<code>-mshort-offset</code>	<code>-mpointer16</code>
<code>-mshort-offset</code>	<code>-mpointer16</code> and <code>-mshort-offset</code>

**Example of compiling:**

```
xgcc -B$(TOOL_DIR)/ -c -O -gstabs -fno-builtin
-I$(TOOL_DIR)/include -mpointer16 main.c
```

When the `-c` option is specified, the `-mpointer16` option will also be passed to the assembler.

**Example of assembling:**

```
xgcc -B$(TOOL_DIR)/ -c -xassembler-with-cpp -Wa,--gstabs
-Wa,-mpointer16 boot.s
```

After `boot.s` is processed in the preprocessor, it will be assembled with the `-mpointer16` option specified.

## 6.4 Compiler Output

This section explains the assembly sources output by the **xgcc** C compiler and the registers used by the **xgcc**.

### 6.4.1 Output Contents

After compiling C sources, the **xgcc** C compiler outputs the following contents:

- S1C17 Core instruction set mnemonics
- Extended instruction mnemonics
- Assembler directives

All but the basic instructions are output using extended instructions.

Since the system control instructions cannot be expressed in the C source, use in-line assemble by `asm` or an assembly source file to process them.

Example: `asm("halt");`

Assembler directives are output for section and data definitions.

The following describes the sections where instructions and data are set.

#### Instructions

All instructions are located in the `.text` section.

#### Constants

Constants are located in the `.rodata` section.

```
Example: const int i=1;          .global   i
                                .section   .rodata
                                .align     2
                                .type      i,@object
                                .size      i,4
                                i:
                                .long     1
```

#### Global and static variables with initial values

These variables are located in the `.data` section.

```
Example: int i=1;               .global   i
                                .section   .data
                                .align     2
                                .type      i,@object
                                .size      i,4
                                i:
                                .long     1
```

#### Global and static variables without initial values

These variables are located in the `.bss` section.

```
Example: int i;                 .global   i
                                .section   .bss
                                .align     2
                                .type      i,@object
                                .size      i,4
                                i:
                                .zero     4
```

For all symbols including function names and labels, symbol information by the `.stab` assembler directive is inserted (when the `-gstabs` option is specified).

## 6.4.2 Data Representation

The `xgcc` C compiler supports all data types under ANSI C. Table 6.4.2.1 below lists the size of each type (in bytes) and the effective range of numeric values that can be expressed in each type.

Table 6.4.2.1 Data type and size

Data type	Size	Effective range of a number
<code>char</code>	1	-128 to 127
<code>unsigned char</code>	1	0 to 255
<code>short</code>	2	-32768 to 32767
<code>unsigned short</code>	2	0 to 65535
<code>int</code>	2	-32768 to 32767
<code>unsigned int</code>	2	0 to 65535
<code>long</code>	4	-2147483648 to 2147483647
<code>unsigned long</code>	4	0 to 4294967295
<code>long long int</code>	8	-9223372036854775808 to 9223372036854775807
<code>unsigned long long int</code>	8	0 to 18446744073709551615
<code>pointer</code>	4	0 to 16777215
<code>float</code>	4	1.175e-38 to 3.403e+38 (normalized number)
<code>double</code>	8	2.225e-308 to 1.798e+308 (normalized number)
<code>long double</code>	8	2.225e-308 to 1.798e+308 (normalized number)
<code>long long</code>	8	-9223372036854775808 to 9223372036854775807
<code>unsigned long long</code>	8	0 to 18446744073709551615
<code>wchar_t</code>	2	0 to 65535

The `float` and `double` types conform to the IEEE standard format.

Handling of `long long`-type constants requires the suffix `LL` or `ll` (`long long` type) or `ULL` or `ull` (`unsigned long long` type). If this suffix is not present, an error is assumed, since the compiler may not be able to recognize `long long`-type constants as such.

Example: `long long ll_val;`

```
ll_val = 0x1234567812345678;
```

```
→error: integer constant is too large for "long" type
```

```
ll_val = 0x1234567812345678LL;
```

```
→OK
```

Type `wchar_t` is the data type needed to handle wide characters. This data type is defined in `stdlib.h/std-def.h` as the type `unsigned short`.

### Store positions in memory

The positions in the memory where data is stored depend on the data type. The `short` and `int` type variables are aligned at 2-byte boundary addresses, and the `long` and `double` type variables are aligned at 4-byte boundary addresses.

### Structure data

Structure data is located in the memory beginning with a 4-byte boundary address. Members are located in the memory according to the size of each data type in the order they are defined.

The following shows an example of how structure is defined, and where it is located.

Example: `struct Sample {`

```
    char    cData;
    short   sData;
    char    cArray[3];
    long    lData;
};
```

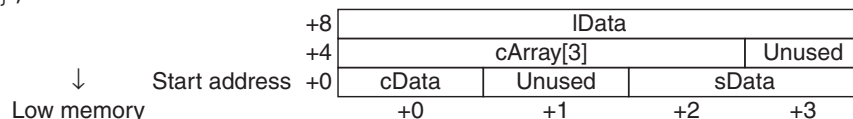


Figure 6.4.2.1 Sample locations of structure data in the memory

As shown in the diagram above, some unused areas may remain in the memory depending on the data type of a member.

## Accessing bit fields

Bit fields with an 8-bit or less bit width will be accessed in byte size as shown below due to the compiler optimization processing even if the bit field is defined as an `unsigned short` type. Take this into consideration when accessing a device or I/O memory that needs to be accessed in 16-bit size.

### Program

```

struct IFTag {
    volatile union {
        volatile struct {
            unsigned short DATA    : 1;        /* 1bit */
            unsigned short Dummy    : 15;
        } bCTL;
        unsigned short usCTL;
    } rOUT;
};

struct g_GAtag {
    volatile struct IFTag    g_IF;
};

volatile struct g_GAtag *pg_GAtag;
main()
{
    pg_GAtag = (struct g_GAtag *)0x8300;
    pg_GAtag->g_IF.rOUT.bCTL.DATA = 1;    (*)
    return;
}

```

### Code compiled from the source line (\*)

```

pg_GAtag->g_IF.rOUT.bCTL.DATA = 1;

ld.b    %r3, [%r2]    ; Byte access
or      %r3, 0x1
ld.b    [%r2], %r3    ; Byte access

```

### 6.4.3 Method of Using Registers

The following shows how the **xgcc** C compiler uses general-purpose registers.

Table 6.4.3.1 Method of using general-purpose registers by **xgcc**

Register	Method of use
%r0	Register for passing argument (1st word) Register for storing returned values (8/16-bit data, pointer, 16 low-order bits of 32-bit data)
%r1	Register for passing argument (2nd word) Register for storing returned values (16 high-order bits of 32-bit data)
%r2	Register for passing argument (3rd word)
%r3	Register for passing argument (4th word)
%r4	Registers that need have to their values saved when calling a function
%r5	
%r6	
%r7	

#### Registers for passing arguments (%r0 to %r3)

These registers are used to store arguments when calling a function. Arguments exceeding four words are stored in the stack before being passed. They are used as scratch registers before storing arguments.

%r0 ← First argument  
 %r1 ← Second argument  
 %r2 ← Third argument  
 %r3 ← Fourth argument

A pair of the registers is used to store a 32-bit (long) argument.

%r1 (high-order 16 bits) and %r0 (low-order 16 bits)  
 %r3 (high-order 16 bits) and %r2 (low-order 16 bits)

Examples:

- First argument: long, second argument: long  
`foo( long lData1, long lData2 );`  
 %r0 ← lData1 (low-order 16 bits)  
 %r1 ← lData1 (high-order 16 bits)  
 %r2 ← lData2 (low-order 16 bits)  
 %r3 ← lData2 (high-order 16 bits)
- First argument: short, second argument: long  
`foo( short sData, long lData );`  
 %r0 ← sData (16 bits)  
 %r1 Unused  
 %r2 ← lData (low-order 16 bits)  
 %r3 ← lData (high-order 16 bits)
- First argument: long, second argument: short, third argument: short  
`foo( long lData, short sData1, short sData2 );`  
 %r0 ← lData (low-order 16 bits)  
 %r1 ← lData (high-order 16 bits)  
 %r2 ← sData1 (16 bits)  
 %r3 ← sData2 (16 bits)

64-bit arguments (long long, double or long double type) are saved to the stack and passed to functions.

If the return value is a 64-bit data (long long, double or long double type), the memory area for storing the return value is allocated and its start address is loaded to %r0 before calling the function.

## Registers for storing returned values (%r0, %r1)

These registers are used to store returned values. They are used as scratch registers before storing a returned value.

- When the returned value is an 8-bit/16-bit data or a pointer (24 bits)
  - %r0 ← Returned value
  - %r1 Unused
- When the returned value is a 32-bit data
  - %r0 ← Returned value (low-order 16 bits)
  - %r1 ← Returned value (high-order 16 bits)

## Registers for saving values when calling a function (%r4 to %r7)

These registers are used to store the calculation results of expressions and local variables. These register values after returning from a function must be the same as those when the function was called. Therefore, the called function has to save and restore the register values if it modifies the register contents.

### 6.4.4 Function Call

#### The way arguments are passed

When calling a function, up to four arguments are stored in registers for passing argument (%r0 to %r3) while larger arguments are stored in the stack frame of the calling function (explained in the next section) before they are passed.

#### Handling of structure arguments

When an argument is a 64-bit or smaller structure, the values of the structure members are stored in the registers (%r0 to %r3) to pass through the function if the registers can be used. If the registers for passing argument (%r0 to %r3) cannot be used, the values of the structure members are passed through the stack.

When an argument is a structure larger than 64 bits, the values of the structure members are passed through the stack.

### 6.4.5 Stack Frame

When calling a function, the **xgcc** C compiler creates the stack frame shown in Figure 6.4.5.1. The start address of the stack frame is always a 32-bit boundary address.

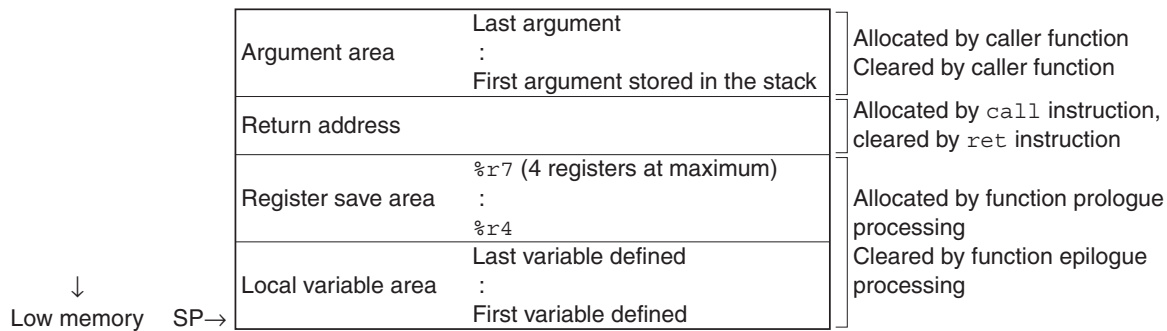


Figure 6.4.5.1 Stack frame

#### Argument area

If there are any arguments for function call that cannot be stored in the registers for passing argument, an area is allocated in the stack frame. All arguments are located at 4-byte boundaries.

#### Return address

This is the return address to the caller function.



### Register save area

If any registers from %r4 to %r7 are used by the caller function, they are saved to this area.

If none of the registers from %r4 to %r7 is used by the caller function, this area is not allocated.

### Local variable area

If there are any local variables defined in the called function that cannot be stored in registers, an area is allocated in the stack frame. Then they are saved sequentially beginning with the last-declared variable at boundary addresses according to the data types.

Example: {

```

char   cData;
short  sData;
long   lData;
      :
```

}

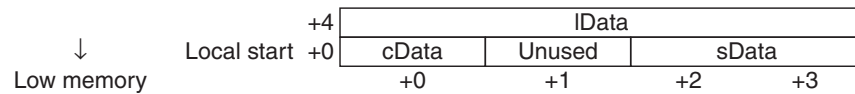


Figure 6.4.5.2 Example of local variables saved to stack

Depending on the source codes, the variables may not be located in the order of declarations due to optimization.

This area is not allocated if there is no local variable that needs to be saved in the stack.

## 6.4.6 Grammar of C Source

Refer to Section 4.2, "Grammar of C Source", for data type, library functions and header files, in-line assemble, and prototype declarations (declaring interrupt handler functions).

## 6.5 Functions of **xgcc** and Usage Precautions

---

- For details about the **xgcc** C compiler, refer to the documents for the gnu compiler.  
The documents can be acquired from the GNU mirror sites located in various places around the world through Internet, etc.
- The **xgcc** C compiler does not support declaration of user sections using "`__attribute__.`"  
`<Type><Function name> __attribute__ ((section "<Section name>"));`  
To locate a function at a specific address, generate and link the object file that includes the function only.
- The `#pragma` preprocessor directive is not guaranteed to work properly. Use it at your own risk.

# 7 Library

This chapter explains the emulation library and the ANSI library included in the S1C17 Family C Compiler Package.

## 7.1 Library Overview

---

Briefly described below are general aspects of the libraries supplied with this package.

### 7.1.1 Library Files

The libraries comprise the following components:

<b>libc.a</b>	ANSI library Provides ANSI standard functions.
<b>libgcc.a</b>	Emulation library Provides single-precision (32-bit) and double-precision (64-bit) floating-point functions including arithmetic operations, comparison and type conversion, integer multiplication/division/sift functions, and long long-type addition/subtraction functions.
<b>libstdio.a</b>	Simulated I/O library Provides a library initialize function (see Section 7.3.3) and lower-level functions for inputs/outputs (see Section 7.3.4).

The libraries are installed in the following separate directories for each memory model.

```
\EPSON
  \gnu17
    \lib
      \24bit 24-bit (16MB) memory model libraries
        libc.a
        libgcc.a
        libstdio.a
      \16bit 16-bit (64KB) memory model libraries
        libc.a
        libgcc.a
        libstdio.a
```

Link the 16-bit libraries with the application program when the `-mpointer16` option is specified in the C compiler and assembler.

## 7.1.2 Precautions to Be Taken When Adding a Library

There is a dependency relationship between the libraries.

When writing to a \*.mak or \*.lds file, specify the libraries in the sequence below.

1. Additional libraries

2. libc.a

3. libgcc.a

4. libc.a (Duplication with 2 does not cause an error. Both files can be referenced normally.)

The object file (or library) can reference only the files present after it, in the order in which they are passed to the linker. If the added library is specified last, none of the external libraries can be used in the added library. Because the basic functions such as float and double arithmetic and the ANSI library cannot be used, always make sure the added library is located before the emulation and ANSI libraries.

Example:

1. NG

```
ld.exe -T withmylib.lds -o withmylib.elf boot.o libc.a libgcc.a libc.a mylib.a
```

If mylib.a is using the emulation and ANSI libraries, an error should always occur during linking.

2. OK

```
ld.exe -T withmylib.lds -o withmylib.elf boot.o mylib.a libc.a libgcc.a libc.a
```

No errors should occur during linking, allowing mylib.a to use the emulation and ANSI libraries normally.

If the added libraries have a dependent relationship, make sure the basic library is located last.

Example:

lib1.a calls only the emulation and ANSI libraries

lib2.a calls lib1.a in addition to the emulation and ANSI libraries

lib3.a calls lib1.a and lib2.a in addition to the emulation and ANSI libraries

```
ld.exe -T withmylib.lds -o withmylib.elf boot.o lib3.a lib2.a lib1.a libc.a libgcc.a
libc.a
```

Refer to Section 5.7.4, "Setting Linker Options", for how to add libraries using the **IDE**.

## 7.2 Emulation Library

---

### 7.2.1 Overview

The S1C17 Family C Compiler Package contains an emulation library (`libgcc.a`) that supports the arithmetic operation, comparison, and type conversion of single-precision (32-bit) and double-precision (64-bit) floating-point numbers that conform to IEEE format, integer multiplication/division/sift operations, and `long long`-type addition/subtraction. The `xgcc` C compiler calls up functions from this library when a floating-point number, `long long` data or integer calculation is performed. Since library functions exchange data via a designated general-purpose register/stack, they can be called from an assembly source. To use emulation library functions, specify `libgcc.a` or `libc.a` during linkage.

#### Registers used in the libraries

- The registers `%r0` to `%r7` are used.
- The registers `%r4` to `%r7` are protected by saving to the stack before execution of a function and by restoring from the stack after completion of the function.

## 7.2.2 Floating-point Calculation Functions

### Function list

Table 7.2.2.1 below lists the floating-point calculation functions.

Table 7.2.2.1 Floating-point calculation functions

Classification	Function name		Functionality
Double-precision floating-point calculation	<code>__adddf3</code>	Addition	$x \leftarrow a + b$
	<code>__subdf3</code>	Subtraction	$x \leftarrow a - b$
	<code>__muldf3</code>	Multiplication	$x \leftarrow a * b$
	<code>__divdf3</code>	Division	$x \leftarrow a / b$
	<code>__negdf2</code>	Sign inversion	$x \leftarrow -a$
Single-precision floating-point calculation	<code>__addsf3</code>	Addition	$x \leftarrow a + b$
	<code>__subsf3</code>	Subtraction	$x \leftarrow a - b$
	<code>__mulsf3</code>	Multiplication	$x \leftarrow a * b$
	<code>__divsf3</code>	Division	$x \leftarrow a / b$
	<code>__negsf2</code>	Sign inversion	$x \leftarrow -a$
Type conversion	<code>__fixunsdfsi</code>	double $\rightarrow$ unsigned int	$x \leftarrow a$
	<code>__fixdfsi</code>	double $\rightarrow$ int	$x \leftarrow a$
	<code>__floatsidf</code>	int $\rightarrow$ double	$x \leftarrow a$
	<code>__fixunssfsi</code>	float $\rightarrow$ unsigned int	$x \leftarrow a$
	<code>__fixsfsi</code>	float $\rightarrow$ int	$x \leftarrow a$
	<code>__floatsisf</code>	int $\rightarrow$ float	$x \leftarrow a$
	<code>__truncdfsf2</code>	double $\rightarrow$ float	$x \leftarrow a$
	<code>__extendsfdf2</code>	float $\rightarrow$ double	$x \leftarrow a$
Double-precision floating-point comparison	<code>__fcmpd</code>	Comparison of double type	PSR change $\leftarrow a - b$
	<code>__eqdf2</code>	Comparison of double type ( $a = b$ )	PSR change $\leftarrow a - b, x \leftarrow 1 \mid 0^*$
	<code>__nedf2</code>	Comparison of double type ( $a \neq b$ )	PSR change $\leftarrow a - b, x \leftarrow 1 \mid 0^*$
	<code>__gtdf2</code>	Comparison of double type ( $a > b$ )	PSR change $\leftarrow a - b, x \leftarrow 1 \mid 0^*$
	<code>__gedf2</code>	Comparison of double type ( $a \geq b$ )	PSR change $\leftarrow a - b, x \leftarrow 1 \mid 0^*$
	<code>__ltdf2</code>	Comparison of double type ( $a < b$ )	PSR change $\leftarrow a - b, x \leftarrow 1 \mid 0^*$
	<code>__ledf2</code>	Comparison of double type ( $a \leq b$ )	PSR change $\leftarrow a - b, x \leftarrow 1 \mid 0^*$
Single-precision floating-point comparison	<code>__fcmps</code>	Comparison of float type	PSR change $\leftarrow a - b$
	<code>__eqsf2</code>	Comparison of float type ( $a = b$ )	PSR change $\leftarrow a - b, x \leftarrow 1 \mid 0^*$
	<code>__nesf2</code>	Comparison of float type ( $a \neq b$ )	PSR change $\leftarrow a - b, x \leftarrow 1 \mid 0^*$
	<code>__gtsf2</code>	Comparison of float type ( $a > b$ )	PSR change $\leftarrow a - b, x \leftarrow 1 \mid 0^*$
	<code>__gesf2</code>	Comparison of float type ( $a \geq b$ )	PSR change $\leftarrow a - b, x \leftarrow 1 \mid 0^*$
	<code>__ltsf2</code>	Comparison of float type ( $a < b$ )	PSR change $\leftarrow a - b, x \leftarrow 1 \mid 0^*$
	<code>__lesf2</code>	Comparison of float type ( $a \leq b$ )	PSR change $\leftarrow a - b, x \leftarrow 1 \mid 0^*$

\*  $x = 1$  if true,  $x = 0$  if false

- If the operation resulted in an overflow or underflow, infinity or negative infinity (see next section) is returned.
- The comparison function changes the C, V, Z or N flag of the PSR depending on the result of  $op1 - op2$ , as shown below. Other flags are not changed.

Comparison result	C	V	Z	N
$op1 > op2$	0	0	0	0
$op1 = op2$	0	0	1	0
$op1 < op2$	1	0	0	1

- During type conversion, values are rounded.

## Floating-point format

The `xgcc` C compiler supports the `float` type (32-bit single-precision) and the `double` type (64-bit double-precision) floating-point numbers conforming to IEEE standards.

The following shows the internal format of floating-point numbers.

### Format of double-precision floating-point number

The real number of the `double` type consists of 64 bits, as shown below.



Bit 63: Sign bit (1 bit)

Bits 62–52: Exponent part (11 bits)

Bits 51–0: Fixed-point part (52 bits)

The result of a floating-point calculation is stored in the 64-bit area beginning with the address loaded in the `%r0` register.

The following shows the relationship of the effective range, floating-point representation, and internal data of the `double` type.

+0:	0.0e+0	0x00000000	00000000
-0:	-0.0e+0	0x80000000	00000000
Maximum normalized number:	1.79769e+308	0x7fefffff	ffffffff
Minimum normalized number:	2.22507e-308	0x00100000	00000000
Maximum unnormalized number:	2.22507e-308	0x000fffff	ffffffff
Minimum unnormalized number:	4.94065e-324	0x00000000	00000001
Infinity:		0x7ff00000	00000000
Negative infinity:		0xfff00000	00000000

Values `0x7ff00000 00000001` to `0x7fffffff ffffffff` and `0xfff00000 00000001` to `0xffffffff ffffffff` are not recognized as numeric values.

### Format of single-precision floating-point number

The real number of the `float` type consists of 32 bits, as shown below.



Bit 31: Sign bit (1 bit)

Bits 30–23: Exponent part (8 bits)

Bits 22–0: Fixed-point part (23 bits)

This type of value occupies two registers. For example, the result of a floating-point calculation is stored in the `%r1` and `%r0` registers.

`%r1` register: Sign bit, exponent part, and 7 high-order bits of fixed-point part (22:16)

`%r0` register: 16 low-order bits of fixed-point part (15:0)

The following shows the relationship of the effective range, floating-point representation, and internal data of the `float` type.

+0:	0.0e+0f	0x00000000
-0:	-0.0e+0f	0x80000000
Maximum normalized number:	3.40282e+38f	0x7f7fffff
Minimum normalized number:	1.17549e-38f	0x00800000
Maximum unnormalized number:	1.17549e-38f	0x007fffff
Minimum unnormalized number:	1.40129e-45f	0x00000001
Infinity:		0x7f800000
Negative infinity:		0xff800000

Values `0x7f800001` to `0x7fffffff` and `0xff800001` to `0xffffffff` are not recognized as numeric values.

### Note

The floating-point numbers in the `xgcc` C compiler differ from the IEEE-based FPU in precision and functionality, including the manner in which infinity is handled.

## 7.2.3 Integral Calculation Functions

Table 7.2.3.1 below lists the integral calculation functions.

Table 7.2.3.1 Integral calculation functions

Classification	Function name	Functionality	
Integral calculation	<code>__divsi3</code>	Signed 32-bit integral division	$x \leftarrow a / b$
	<code>__modsi3</code>	Signed 32-bit remainder calculation	$x \leftarrow a \% b$
	<code>__udivsi3</code>	Unsigned 32-bit integral division	$x \leftarrow a / b$
	<code>__umodsi3</code>	Unsigned 32-bit remainder calculation	$x \leftarrow a \% b$
	<code>__mulsi3</code>	32-bit multiplication	$x \leftarrow a * b$
	<code>__divhi3</code>	Signed 16-bit integral division	$x \leftarrow a / b$
	<code>__modhi3</code>	Signed 16-bit remainder calculation	$x \leftarrow a \% b$
	<code>__udivhi3</code>	Unsigned 16-bit integral division	$x \leftarrow a / b$
	<code>__umodhi3</code>	Unsigned 16-bit remainder calculation	$x \leftarrow a \% b$
	<code>__mulhi3</code>	16-bit multiplication	$x \leftarrow a / b$
Integral shift	<code>__ashlsi3</code>	32-bit arithmetical shift to left	$x \leftarrow a \ll b$ bits
	<code>__ashrsi3</code>	32-bit arithmetical shift to right	$x \leftarrow a \gg b$ bits
	<code>__lshrsi3</code>	32-bit logical shift to right	$x \leftarrow a \gg b$ bits
	<code>__ashlhi3</code>	16-bit arithmetical shift to left	$x \leftarrow a \ll b$ bits
	<code>__ashrhi3</code>	16-bit arithmetical shift to right	$x \leftarrow a \gg b$ bits
	<code>__lshrhi3</code>	16-bit logical shift to right	$x \leftarrow a \gg b$ bits

## 7.2.4 long long Type Calculation Functions

Table 7.2.4.1 below lists the long long type calculation functions.

Table 7.2.4.1 long long type calculation functions

Classification	Function name	Functionality	
long long type calculation	<code>__addi3</code>	Signed 64-bit addition	$x \leftarrow a + b$
	<code>__subdi3</code>	Signed 64-bit subtraction	$x \leftarrow a - b$
	<code>__muldi3</code>	Signed 64-bit multiplication	$x \leftarrow a * b$
	<code>__divdi3</code>	Signed 64-bit division	$x \leftarrow a / b$
	<code>__udivdi3</code>	Unsigned 64-bit division	$x \leftarrow a / b$
	<code>__moddi3</code>	Signed 64-bit remainder calculation	$x \leftarrow a \% b$
	<code>__umoddi3</code>	Unsigned 64-bit remainder calculation	$x \leftarrow a \% b$
long long type shift	<code>__lshrdi3</code>	64-bit logical shift to right	$x \leftarrow a \gg b$ bits
	<code>__ashldi3</code>	64-bit arithmetical shift to left	$x \leftarrow a \ll b$ bits
	<code>__ashrdi3</code>	64-bit arithmetical shift to right	$x \leftarrow a \gg b$ bits
Type conversion	<code>__fixunsdfdi</code>	double $\rightarrow$ unsigned long long	$x \leftarrow a$
	<code>__fixdfdi</code>	double $\rightarrow$ long long	$x \leftarrow a$
	<code>__floatdidf</code>	long long $\rightarrow$ double	$x \leftarrow a$
	<code>__fixunssfdi</code>	float $\rightarrow$ unsigned long long	$x \leftarrow a$
	<code>__fixsfdi</code>	float $\rightarrow$ long long	$x \leftarrow a$
	<code>__floatdisf</code>	long long $\rightarrow$ float	$x \leftarrow a$
long long type comparison	<code>__cmpdi2</code>	Comparison (long long)	$x \leftarrow 2   1   0$ *1
	<code>__ucmpdi2</code>	Comparison (unsigned long long)	$x \leftarrow 2   1   0$ *1
Other	<code>__ffsdi2</code>	Bit scan	$x \leftarrow 64$ to 0 *2

\*1 The long long comparison function returns the following values according to the result of  $op1 - op2$ .

$op1 > op2 \rightarrow 2$

$op1 = op2 \rightarrow 1$

$op1 < op2 \rightarrow 0$

\*2 Bits are scanned for logic 1 beginning with the LSB, and the position of the first bit found with the value 1 is returned.

If the first bit with the value 1 is the LSB: 1

If the first bit with the value 1 is the MSB: 64

If no bits are found with the value 1: 0



## 7.3 ANSI Library

### 7.3.1 Overview

The S1C17 Family C Compiler Package contains an ANSI library.

Each function in this library has ANSI-standard functionality. However, some file-related functions are dummy functions due to embedded microcomputer specifications.

The `libc.a` ANSI library file is installed in separate directories (`\gnu17\lib\24bit` and `\gnu17\lib\16bit`) for each memory model. A long long-type ANSI library is included in `libgcc.a`.

The following header files which contain definitions of each function are installed in the `include` directory.

`stdio.h` `stdlib.h` `time.h` `math.h` `errno.h` `float.h` `limits.h` `ctype.h`  
`string.h` `stdarg.h` `setjmp.h` `smcvals.h` `stddef.h`

#### Registers used in the library

- The registers `%r0` to `%r7` are used.
- The registers `%r4` to `%r7` are protected by saving to the stack before execution of a function and by restoring from the stack after completion of the function.

### 7.3.2 ANSI Library Function List

The contents of the Reentrant column in the tables are as follows:

Reentrant: Reentrant function

Nonreentrant: Non-reentrant function

Dummy: Dummy function (It must be modified according to your system.)

Conditional: Non-reentrant function (This function refers to a global variable or it calls a dummy function. It can be used as a reentrant function if there is no change in the global variable, and your created `read()` and `write()` are reentrant functions.)

#### Input/output functions

The table below lists the input/output functions included in `libc.a`.

Table 7.3.2.1 Input/output functions

Header file: `stdio.h`

Function	Functionality	Reentrant	Notes
<code>FILE *fopen(char *filename, char *mode);</code>	Dummy	Dummy	
<code>FILE *freopen(char *filename, char *mode, FILE *stream);</code>	Dummy	Dummy	
<code>int fclose(FILE *stream);</code>	Dummy	Dummy	
<code>int fflush(FILE *stream);</code>	Dummy	Dummy	
<code>int fseek(FILE *stream, long int offset, int origin);</code>	Dummy	Dummy	
<code>long int ftell(FILE *stream);</code>	Dummy	Dummy	
<code>void rewind(FILE *stream);</code>	Dummy	Dummy	
<code>int fgetpos(FILE *stream, fpos_t *ptr);</code>	Dummy	Dummy	
<code>int fsetpos(FILE *stream, fpos_t *ptr);</code>	Dummy	Dummy	
<code>size_t fread(void *ptr, size_t size, size_t count, FILE *stream);</code>	Input array element from <code>stdin</code> .	Conditional	Refer to global variables <code>stdin</code> and <code>_iob</code> , and call <code>read</code> function.
<code>size_t fwrite(void *ptr, size_t size, size_t count, FILE *stream);</code>	Output array element to <code>stdout</code> .	Conditional	Refer to global variables <code>stdout</code> , <code>stderr</code> and <code>_iob</code> , and call <code>write</code> function.
<code>int fgetc(FILE *stream);</code>	Input one character from <code>stdin</code> .	Conditional	Refer to global variables <code>stdin</code> and <code>_iob</code> , and call <code>read</code> function.
<code>int getc(FILE *stream);</code>	Input one character from <code>stdin</code> .	Conditional	Refer to global variables <code>stdin</code> and <code>_iob</code> , and call <code>read</code> function.
<code>int getchar( );</code>	Input one character from <code>stdin</code> .	Conditional	Refer to global variables <code>stdin</code> and <code>_iob</code> , and call <code>read</code> function.

## 7 LIBRARY

Function	Functionality	Reentrant	Notes
<code>int ungetc(int c, FILE *stream);</code>	Push one character back to input buffer.	Nonreentrant	Refer to global variables <code>stdin</code> , <code>stdout</code> , <code>stderr</code> , and <code>_iob</code> , returned value overwrite.
<code>char *fgets(char *s, int n, FILE *stream);</code>	Input character string from <code>stdin</code> .	Conditional	Refer to global variables <code>stdin</code> and <code>_iob</code> , and call <code>read</code> function.
<code>char *gets(char *s);</code>	Input character string from <code>stdin</code> .	Conditional	Refer to global variables <code>stdin</code> and <code>_iob</code> , and call <code>read</code> function.
<code>int fputc(int c, FILE *stream);</code>	Output one character to <code>stdout</code> .	Conditional	Refer to global variables <code>stdout</code> , <code>stderr</code> and <code>_iob</code> , and call <code>write</code> function.
<code>int putc(int c, FILE *stream);</code>	Output one character to <code>stdout</code> .	Conditional	Refer to global variables <code>stdout</code> , <code>stderr</code> and <code>_iob</code> , and call <code>write</code> function.
<code>int putchar(int c);</code>	Output one character to <code>stdout</code> .	Conditional	Refer to global variables <code>stdout</code> , <code>stderr</code> and <code>_iob</code> , and call <code>write</code> function.
<code>int fputs(char *s, FILE *stream);</code>	Output character string to <code>stdout</code> .	Conditional	Refer to global variables <code>stdout</code> , <code>stderr</code> and <code>_iob</code> , and call <code>write</code> function.
<code>int puts(char *s);</code>	Output character string to <code>stdout</code> .	Conditional	Refer to global variables <code>stdout</code> , <code>stderr</code> and <code>_iob</code> , and call <code>write</code> function.
<code>int remove(char *filename);</code>	Dummy	Dummy	
<code>int rename(char *oldname, char *newname);</code>	Dummy	Dummy	
<code>void setbuf(FILE *stream, char *buf);</code>	Dummy	Dummy	
<code>int setvbuf(FILE *stream, char *buf, int type, size_t size);</code>	Dummy	Dummy	
<code>FILE *tmpfile( );</code>	Dummy	Dummy	
<code>char *tmpnam(char *buf);</code>	Dummy	Dummy	
<code>int feof(FILE *stream);</code>	Dummy	Dummy	
<code>int ferror(FILE *stream);</code>	Dummy	Dummy	
<code>void clearerr(FILE *stream);</code>	Dummy	Dummy	
<code>void perror(char *s);</code>	Output error information to <code>stdout</code> .	Nonreentrant	Refer to global variables <code>stdout</code> and <code>_iob</code> , change <code>errno</code> , and call <code>read</code> function.
<code>int fscanf(FILE *stream, char *format, ...);</code>	Input from <code>stdin</code> with format specified.	Nonreentrant	Refer to global variables <code>stdout</code> and <code>_iob</code> , change <code>errno</code> , and call <code>read</code> function.
<code>int scanf(char *format, ...);</code>	Input from <code>stdin</code> with format specified.	Nonreentrant	Refer to global variables <code>stdout</code> and <code>_iob</code> , change <code>errno</code> , and call <code>read</code> function.
<code>int sscanf(char *s, char *format, ...);</code>	Input from character string with format specified.	Nonreentrant	Change global variable <code>errno</code> .
<code>int fprintf(FILE *stream, char *format, ...);</code>	Output to <code>stdout</code> with format specified.	Conditional	Refer to global variables <code>stdout</code> , <code>stderr</code> and <code>_iob</code> , and call <code>write</code> function.
<code>int printf(char *format, ...);</code>	Output to <code>stdout</code> with format specified.	Conditional	Refer to global variables <code>stdout</code> , <code>stderr</code> and <code>_iob</code> , and call <code>write</code> function.
<code>int sprintf(char *s, char *format, ...);</code>	Output to array with format specified.	Reentrant	
<code>int vfprintf(FILE *stream, char *format, va_list arg);</code>	Output conversion result to <code>stdout</code> .	Conditional	Refer to global variables <code>stdout</code> , <code>stderr</code> and <code>_iob</code> , and call <code>write</code> function.
<code>int vprintf(char *s, va_list arg);</code>	Output conversion result to <code>stdout</code> .	Conditional	Refer to global variables <code>stdout</code> , <code>stderr</code> and <code>_iob</code> , and call <code>write</code> function.
<code>int vsprintf(char *s, char *format, va_list arg);</code>	Output conversion result to array.	Reentrant	

**Note:** The file system is disabled; `stdin` and `stdout` are enabled. When using `stdin` and `stdout`, the `read()` and `write()` functions are needed, respectively. Refer to Section 7.3.4 for more information.

## Utility functions

The table below lists the utility functions included in `libc.a`.

Table 7.3.2.2 Utility functions

Header file: `stdlib.h`

Function	Functionality	Reentrant	Notes
<code>void *malloc(size_t size);</code>	Allocate area.	Nonreentrant	Change global variables <code>errno</code> , <code>ansi_ucStartAlloc</code> , <code>ansi_ucEndAlloc</code> , <code>ansi_ucNextAlcP</code> , <code>ansi_ucTblPtr</code> , and <code>ansi_ulRow</code> .
<code>void *calloc(size_t elt_count, size_t elt_size);</code>	Allocate array area.	Nonreentrant	Invalid for call from memory allocate.
<code>void free(void *ptr);</code>	Clear area.	Nonreentrant	Invalid for call from memory allocate.
<code>void *realloc(void *ptr, size_t size);</code>	Change area size.	Nonreentrant	Invalid for call from memory allocate.
<code>int system(char *command);</code>	Dummy	Dummy	
<code>void exit(int status);</code>	Terminate program normally.	Reentrant	Refer to <code>exit</code> , terminates on the side of called later.
<code>void abort( );</code>	Terminate program abnormally.	Reentrant	Refer to <code>exit</code> , terminates on the side of called later.
<code>int atexit(void (*func)(void));</code>	Dummy	Dummy	
<code>char *getenv(char *str);</code>	Dummy	Dummy	
<code>void *bsearch(void *key, void *base, size_t count, size_t size, int (*compare)(void *, void *));</code>	Binary search.	Reentrant	
<code>void qsort(void *base, size_t count, size_t size, int (*compare)(void *, void *));</code>	Quick sort.	Reentrant	
<code>int abs(int x);</code>	Return absolute value (int type).	Reentrant	
<code>long int labs(long int x);</code>	Return absolute value (long type).	Reentrant	
<code>div_t div(int n, int d);</code>	Divide int type.	Nonreentrant	Change global variable <code>errno</code> .
<code>ldiv_t ldiv(int n, int d);</code>	Divide long type.	Nonreentrant	Change global variable <code>errno</code> .
<code>int rand( );</code>	Return pseudo-random number.	Nonreentrant	Change global variable <code>errno</code> .
<code>void srand(unsigned int seed);</code>	Set seed of pseudo-random number.	Nonreentrant	Change global variable <code>errno</code> .
<code>long int atol(char *str);</code>	Convert character string into long type.	Nonreentrant	Change global variable <code>errno</code> .
<code>int atoi(char *str);</code>	Convert character string into int type.	Nonreentrant	Change global variable <code>errno</code> .
<code>double atof(char *str);</code>	Convert character string into double type.	Nonreentrant	Change global variable <code>errno</code> .
<code>double strtod(char *str, char **ptr);</code>	Convert character string into double type.	Nonreentrant	Change global variable <code>errno</code> .
<code>long int strtol(char *str, char **ptr, int base);</code>	Convert character string into long type.	Nonreentrant	Change global variable <code>errno</code> .
<code>unsigned long int strtoul(char *str, char **ptr, int base);</code>	Convert character string into unsigned long type.	Nonreentrant	Change global variable <code>errno</code> .

## Non-local branch functions

The table below lists the non-local branch functions included in `libc.a`.

Table 7.3.2.3 Non-local branch functions

Header file: `setjmp.h`

Function	Functionality	Reentrant	Notes
<code>int setjmp(jmp_buf env);</code>	Non-local branch	Reentrant	
<code>void longjmp(jmp_buf env, int status);</code>	Non-local branch	Reentrant	

## Date and time functions

The table below lists the date and time functions included in `libc.a`.

Table 7.3.2.4 Date and time functions

Header file: `time.h`

Function	Functionality	Reentrant	Notes
<code>clock_t clock( );</code>	Dummy	Dummy	
<code>char *asctime(struct tm *ts);</code>	Dummy	Dummy	
<code>char *ctime(time_t *timeptr);</code>	Dummy	Dummy	
<code>double difftime(time_t t1, time_t t2);</code>	Dummy	Dummy	
<code>struct tm *gmtime(time_t *t);</code>	Convert calendar time to standard time.	Nonreentrant	Change static variable.
<code>struct tm *localtime(time_t *t);</code>	Dummy	Dummy	
<code>time_t mktime(struct tm *tmptr);</code>	Convert standard time to calendar time.	Nonreentrant	Change static variable.
<code>time_t time(time_t *t);</code>	Return current calendar time.	Conditional	Refer to global variable <code>gm_sec</code> .

## Mathematical functions

The table below lists the mathematical functions included in `libc.a`.

Table 7.3.2.5 Mathematical functions

Header files: `math.h`, `errno.h`, `float.h`, `limits.h`

Function	Functionality	Reentrant	Notes
<code>double fabs(double x);</code>	Return absolute value (double type).	Reentrant	
<code>double ceil(double x);</code>	Round up double-type decimal part.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double floor(double x);</code>	Round down double-type decimal part.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double fmod(double x, double y);</code>	Calculate double-type remainder.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double exp(double x);</code>	Exponentiate ( $e^x$ ).	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double log(double x);</code>	Calculate natural logarithm.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double log10(double x);</code>	Calculate common logarithm.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double frexp(double x, int *nptr);</code>	Return mantissa and exponent of floating-point number.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double ldexp(double x, int n);</code>	Return floating-point number from mantissa and exponent.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double modf(double x, double *nptr);</code>	Return integer and decimal parts of floating-point number.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double pow(double x, double y);</code>	Calculate $x^y$ .	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double sqrt(double x);</code>	Calculate square root.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double sin(double x);</code>	Calculate sine.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double cos(double x);</code>	Calculate cosine.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double tan(double x);</code>	Calculate tangent.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double asin(double x);</code>	Calculate arcsine.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double acos(double x);</code>	Calculate arccosine.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double atan(double x);</code>	Calculate arctangent.	Nonreentrant	Destruct ALR and AHR.
<code>double atan2(double y, double x);</code>	Calculate arctangent of $y/x$ .	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double sinh(double x);</code>	Calculate hyperbolic sine.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double cosh(double x);</code>	Calculate hyperbolic cosine.	Nonreentrant	Change global variable <code>errno</code> , destruct ALR and AHR.
<code>double tanh(double x);</code>	Calculate hyperbolic tangent.	Nonreentrant	Destruct ALR and AHR.

## Character functions

The table below lists the character functions included in `libc.a`.

Table 7.3.2.6 Character functions

Header file: `string.h`

Function	Functionality	Reentrant	Notes
<code>char *memchr(char *s, int c, int n);</code>	Return specified character position in the storage area.	Reentrant	
<code>int memcmp(char *s1, char *s2, int n);</code>	Compare storage areas.	Reentrant	
<code>char *memcpy(char *s1, char *s2, int n);</code>	Copy storage area.	Reentrant	
<code>char *memmove(char *s1, char *s2, int n);</code>	Copy the storage area (overlapping allowed).	Reentrant	
<code>char *memset(char *s, int c, int n);</code>	Set character in the storage area.	Reentrant	
<code>char *strcat(char *s1, char *s2);</code>	Concatenate character strings.	Reentrant	
<code>char *strchr(char *s, int c);</code>	Return specified character position found first in the character string.	Reentrant	
<code>int strcmp(char *s1, char *s2);</code>	Compare character strings.	Reentrant	
<code>char *strcpy(char *s1, char *s2);</code>	Copy character string.	Reentrant	
<code>size_t *strcspn(char *s1, char *s2);</code>	Return number of characters from the beginning of the character string until the specified character appears (multiple choices).	Reentrant	
<code>char *strerror(int code);</code>	Return error message character string.	Reentrant	
<code>size_t strlen(char *s);</code>	Return length of character string.	Reentrant	
<code>size_t strncat(char *s1, char *s2, int n);</code>	Concatenate character strings (number of characters specified).	Reentrant	
<code>int strncmp(char *s1, char *s2, int n);</code>	Compare character strings (number of characters specified).	Reentrant	
<code>char *strncpy(char *s1, char *s2, int n);</code>	Copy character string (number of characters specified).	Reentrant	
<code>char *strpbrk(char *s1, char *s2);</code>	Return specified character position (multiple choices) found first in the character string.	Reentrant	
<code>char *strrchr(char *s, int c);</code>	Return specified character position found last in the character string.	Reentrant	
<code>size_t strspn(char *s1, char *s2);</code>	Return number of characters from the beginning of the character string until the non-specified character appears (multiple choices).	Reentrant	
<code>char *strstr(char *s1, char *s2);</code>	Return position where the specified character string appeared first.	Reentrant	
<code>char *strtok(char *s1, char *s2);</code>	Divide the character string into tokens.	Nonreentrant	Change static variable.

## Character type determination/conversion functions

The table below lists the character type determination/conversion functions included in `libc.a`.

Table 7.3.2.7 Character type determination/conversion functions

Header file: `cctype.h`

Function	Functionality	Reentrant
<code>int isalnum(char c);</code>	Determine character type (decimal or alphabet).	Reentrant
<code>int isalpha(char c);</code>	Determine character type (alphabet).	Reentrant
<code>int iscntrl(char c);</code>	Determine character type (control character).	Reentrant
<code>int isdigit(char c);</code>	Determine character type (decimal).	Reentrant
<code>int isgraph(char c);</code>	Determine character type (graphic character).	Reentrant
<code>int islower(char c);</code>	Determine character type (lowercase alphabet).	Reentrant
<code>int isprint(char c);</code>	Determine character type (printable character).	Reentrant
<code>int ispunct(char c);</code>	Determine character type (delimiter).	Reentrant
<code>int isspace(char c);</code>	Determine character type (null character).	Reentrant
<code>int isupper(char c);</code>	Determine character type (uppercase alphabet).	Reentrant
<code>int isxdigit(char c);</code>	Determine character type (hexadecimal).	Reentrant
<code>int tolower(char c);</code>	Convert character type (uppercase alphabet → lowercase).	Reentrant
<code>int toupper(char c);</code>	Convert character type (lowercase alphabet → uppercase).	Reentrant

## Variable argument macros

The table below lists the variable argument macros defined in `stdarg.h`.

Table 7.3.2.8 Variable argument macros

Header file: `stdarg.h`

Macro	Functionality
<code>void va_start(va_list ap, type lastarg);</code>	Initialize the variable argument group.
<code>type va_arg(va_list ap, type);</code>	Return the actual argument.
<code>void va_end(va_list ap);</code>	Return normally from the variable argument function.

### 7.3.3 Declaring and Initializing Global Variables

The ANSI library functions reference the global variables listed in Table 7.3.3.1. These variables have been defined in the `libstdio.a` library. Include "`\include\libstdio.h`" in the C source program and call the `_init_lib()` function to initialize the variables before calling a library function.

For how to initialize the ANSI library using `libstdio.a`, refer to the sample file "`\gnu17\sample\S1C17common\simulator\simulatedIO`".

Table 7.3.3.1 Global variables required of declaration

Global variable	Initial setting	Related header file/function
<b>FILE _iob[FOPEN_MAX +1];</b> FOPEN_MAX=3, defined in <code>stdio.h</code> File structure data for standard input/output streams	<code>_iob[N]._flg = _UGETN;</code> <code>_iob[N]._buf = 0;</code> <code>_iob[N]._fd = N;</code> (N=0-2) <code>_iob[0]: Input data for stdin</code> <code>_iob[1]: Output data for stdout</code> <code>_iob[2]: Output data for stderr</code>	<b>stdio.h, smcvals.h</b> <code>fgets, fread, fscanf, getc, getchar, gets, scanf, ungetc, perror, fprintf, fputs, fwrite, printf,putc, putchar, puts, vfprintf, vprintf</code>
<b>FILE *stdin;</b> Pointer to standard input/output file structure data <code>_iob[0]</code>	<code>stdin = &amp;_iob[0];</code>	<b>stdio.h</b> <code>fgets, fread, fscanf, getc, getchar, gets, scanf, ungetc</code>
<b>FILE *stdout;</b> Pointer to standard input/output file structure data <code>_iob[1]</code>	<code>stdout = &amp;_iob[1];</code>	<b>stdio.h</b> <code>fprintf, fputs, fwrite, printf,putc, putchar, puts, vfprintf, vprintf</code>
<b>FILE *stderr;</b> Pointer to standard input/output file structure data <code>_iob[2]</code>	<code>stderr = &amp;_iob[2];</code>	<b>stdio.h</b> <code>fprintf, fputs, fwrite, printf, perror,putc, putchar, puts, vfprintf, vprintf</code>
<b>int errno;</b> Variable to store error number	<code>errno = 0;</code>	<b>errno.h</b> <code>fopen, freopen, fseek, fsetpos, perror, remove, rename, tmpfile, tmpnam, fprintf, printf, sprintf, vprintf, vfprintf, fscanf, scanf, sscanf, atof, atoi, calloc, div, ldiv, malloc, realloc, strtod, strtol, strtoul, acos, asin, atan2, ceil, cos, cosh, exp, fabs, floor, fmod, frexp, ldexp, log, log10, modf, pow, sin, sinh, sqrt, tan</code>
<b>unsigned int seed;</b> Variable to store seed of random number	<code>seed = 1;</code>	<b>stdlib.h</b> <code>rand, srand</code>
<b>time_t gm_sec;</b> Elapsed time of timer function in seconds from 0:00:00 on January 1, 1970	<code>gm_sec = -1;</code>	<b>time.h</b> <code>time</code>

Among the global variables referenced by the ANSI library functions, those that are used by each function (`malloc`, `calloc`, `realloc`, and `free`) are initialized using the initialization function shown below. This function is defined in `stdlib.h`.

```
int ansi_InitMalloc(unsigned long START_ADDRESS, unsigned long END_ADDRESS);
```

For the `START_ADDRESS` and `END_ADDRESS`, set the start and end addresses of the memory used, respectively. These addresses are adjusted to the 4-byte boundaries within the function.

The following global variables are initialized. These variables are defined in `stdlib.h`.

```
unsigned char *ansi_ucStartAlloc; Pointer to indicate the start address of the heap area
unsigned char *ansi_ucEndAlloc;   Pointer to indicate the end address of the heap area
unsigned char *ansi_ucNxtAlcP;    Address pointer to indicate the beginning of the next new area mapped
unsigned char *ansi_ucTblPtr;     Address pointer to indicate the beginning of the next management area mapped
unsigned long ansi_ulRow;         Line pointer to indicate the next management area mapped
```

Each time storage is reserved for a heap area, eight-byte heap area management data is written from the ending address (`ansi_ucEndAlloc`) toward the beginning address. Be careful to avoid rewriting areas specified as heap areas by the `ansi_InitMalloc()` function by a stack pointer, etc.

\* The `libstdio.a` library does not contain the `ansi_InitMalloc()` function. Be aware that it must be called from the user routine before calling `malloc()` or a similar function. (A heap area cannot be allocated if the `ansi_InitMalloc()` function is not called.)

### 7.3.4 Lower-level Functions

The following three functions (`read`, `write`, and `_exit`) are the lower-level functions called by library functions. A `read` and a `write` function have been defined in the `libstdio.a` library. Before these functions can be used, include "`\include\libstdio.h`" in the C source program and call the `_init_sys()` function. The `_exit` function must be defined in the user program.

For how to use the `libstdio.a`, refer to the sample file "`\gnu17\sample\S1C17common\simulator\simulatedIO`".

#### read function

##### Contents of read function

Format: `int read(int fd, char *buf, int nbytes);`

Argument: `int fd;` File descriptor denoting input  
When called from a library function, 0 (`stdin`) is passed.  
`char *buf;` Pointer to the buffer that stores input data  
`int nbytes;` Number of bytes transferred

Functionality: This function reads up to `nbytes` of data from the user-defined input buffer, and stores it in the buffer indicated by `buf`.

Returned value: Number of bytes actually read from the input buffer  
If the input buffer is empty (EOF) or `nbytes = 0`, 0 is returned.  
If an error occurs, -1 is returned.

Library functions that call the `read` function:

Direct call: `fread`, `getc`, `_doscan` (`_doscan` is a `scanf`-series internal function)

Indirect call: `fgetc`, `fgets`, `getchar`, `gets` (calls `getc`)  
`scanf`, `fscanf`, `sscanf` (calls `_doscan`)

##### Definition of input buffer

Format: `unsigned char READ_BUF[65];` (Variable name is arbitrary; size is fixed to 65 bytes)  
`unsigned char READ_EOF;`

Buffer contents: The size of the input data (1 to max. 64) is stored at the beginning of the buffer (`READ_BUF[0]`). 0 denotes EOF.

The input data is stored in `READ_BUF[1]`, and the following locations.

`READ_EOF` stores the status that indicates whether EOF is reached.

##### Precautions on using a simulated I/O

When using the debugger's simulated I/O, define in the `read` function the global label `READ_FLASH` that is required for the debugger to update the input buffer, then create the function so that new data will be read into the input buffer at that position. (For details about the simulated I/O function, refer to the chapter where the debugger is discussed.)

A `read` function has been defined in the `libstdio.a` library. To use the `read` function, link `libstdio.a` and call the `_init_sys()` function from the boot routine in the user program.



## write function

### Contents of write function

Format: `int write(int fd, char *buf, int nbytes);`

Argument: `int fd;` File descriptor denoting output  
When called from a library function, 1 (stdout) or 2 (stderr) is passed.

`char *buf;` Pointer to the buffer that stores output data

`int nbytes;` Number of transferred bytes

Functionality: The data stored in the buffer indicated by `buf` is written as much as indicated by `nbytes` to the user-defined output buffer.

Returned value: Number of bytes actually written to the output buffer  
If data is written normally, `nbytes` is returned.  
If a write error occurs, a value other than `nbytes` is returned.

Library function that calls the `write` function:

Direct call: `fwrite,putc,_doprint` (`_doprint` is `printf`-series internal function)

Indirect call: `fputc,fputs,putchar,puts` (calls `putc`)  
`printf,fprintf,sprintf,vprintf,vfprintf` (calls `_doprint`)  
`perror` (calls `fprintf`)

### Definition of output buffer

Format: `unsigned char WRITE_BUF[65];`  
(Variable name is arbitrary; size is fixed to 65 bytes)

Buffer content: The size of the output data (1 to max. 64) is stored at the beginning of the buffer (`WRITE_BUF[0]`). 0 denotes EOF.  
The output data is stored in `WRITE_BUF[1]`, and the following locations.

### Precautions on using simulated I/O

When using the debugger's simulated I/O, define in the `write` function the global label `WRITE_FLASH` that is required for the debugger to update the output buffer, and create a function so that data will be output from the output buffer at that position. (For details about the simulated I/O function, refer to the chapter where the debugger is discussed.)

A `write` function has been defined in the `libstdio.a` library. To use the read function, link `libstdio.a` and call the `_init_sys()` function from the boot routine in the user program.

## \_exit function

### Contents of \_exit function

Format: `void _exit(void);`

Functionality: Performs program terminating processing.

Argument: None

Returned value: None

Library function that calls the `_exit` function:  
Direct call: `abort,exit`

THIS PAGE IS BLANK.

# 8 Assembler

This chapter describes the functions of the **as** assembler. For the syntax of the assembly sources, refer to Section 4.3, "Grammar of Assembly Source".

## 8.1 Functions

The **as** assembler assembles (translates) assembly source files that are delivered by the C compiler and creates object files in the machine language. It can also deliver debugging information for purposes of symbolic debugging. This assembler is based on the gnu assembler (as). For details about the **as** assembler, refer to the documents for the gnu assembler. The documents can be acquired from the GNU mirror sites located in various places around the world through Internet, etc.

## 8.2 Input/Output Files

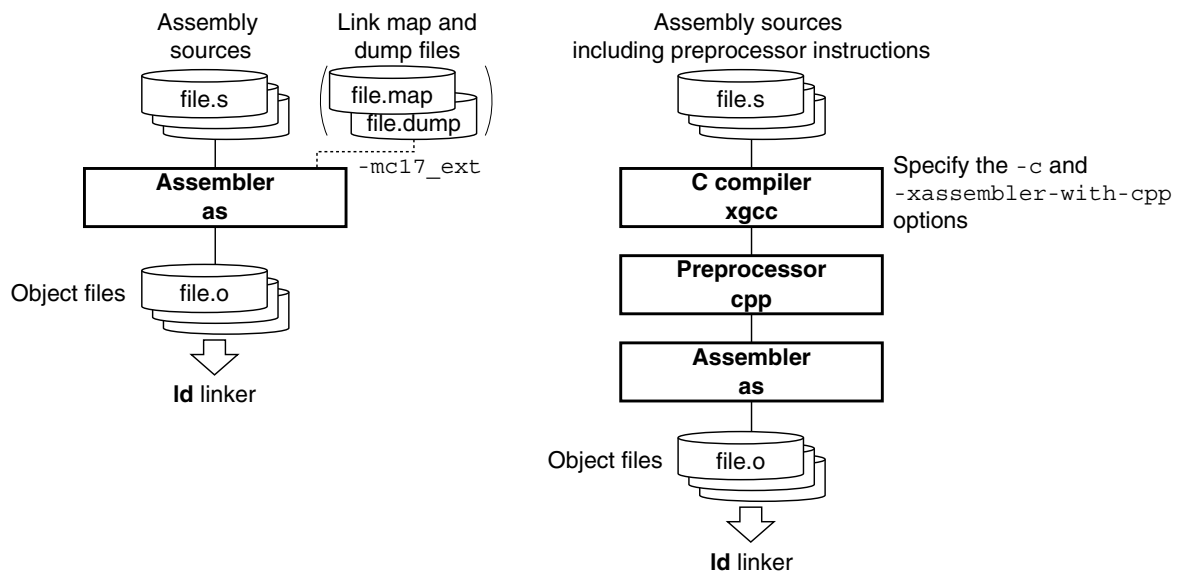


Figure 8.2.1 Flowchart

### 8.2.1 Input Files

#### Assembly source file

File format: Text file

File name: `<file name>.s` (Other extenders than ".s" can be used. A path can also be specified.)

Description: File in which a source program is described. Usually, a file delivered by the **xgcc** C compiler is input there.

If source files were created that only describe basic instructions and assembler directives, they can be input into the **as** assembler directly.

#### Link map file

File format: Text file

File name: `<file name>.map`

Description: File in which object mapping information is described. This file is delivered by the **ld** linker and is used to optimize the code for referencing global symbols when the `-mc17_ext` option is specified.

### Dump file

File format: Text file

File name: *<file name>*.dump

Description: File in which global and local symbol information is described. Use **objdump** to create this file.

```
objdump -t <file name>.elf > <file name>.dump
```

Specify the file name extension as ".dump". This file is used to optimize the code for referencing local symbols when the `-mc17_ext` option is specified.

## 8.2.2 Output File

### Object file

File format: Binary file in elf format

File name: *<file name>*.o (The *<file name>* is the same as that of the input file.)

Description: File in which symbol information and debugging information are added to the program code (machine language).

## 8.3 Starting Method

### 8.3.1 Startup Format

To invoke the **as** assembler, use the command shown below.

**as** *<options>* *<file name>*

*<options>* See Section 8.3.2.

*<file name>* Specify assembly source file name(s) including the extension (.s).

### 8.3.2 Command-line Options

The **as** assembler accepts the gnu assembler standard options. The following lists the principal options only. Refer to the gnu assembler manual for more information.

**-o***<file name>*

Function: **Specify output file name**

Description: This option is used to specify the name of the object file output by the **as** assembler. The *<file name>* must be the same as the input file and input immediately after **-o**.

Default: The default output file name is **a.out**.

**-a** [*<sub-option>*]

Function: **Output assembly list file**

Description: Outputs an assembly list file. The *<sub-option>* controls the output contents.

Example: **-adh1** Requests high-level assembly listing without debugging directives.

Default: No assembly list file is output.

**--gstabs**

Function: **Add debugging information with relative path to source files**

Description: This option is used to creates an output file containing debugging information.

The source file location information is output as a relative path.

Default: No debugging information is output.

In addition to the standard options, the following S1C17 option is available:

**-mpointer16**

Function: **Specify 16-bit pointer mode**

Description: This option is used to generate object files for the 16-bit pointer mode (64KB memory model).

This option just sets a flag to indicate that the 16-bit pointer mode is specified and it does not affect the object code that will be generated.

Default: The assembler generates object files for the 24-bit pointer mode (16MB or 1MB memory model).

**-mc17\_ext** *<dump file name>* *<link map file name>*

Function: **Optimize extended instructions**

Description: This option is used to remove unnecessary **ext** instructions, which were inserted when **s\*/x\*** extended instructions were expanded by the assembler in the first pass, according to the actual distance to each symbol that has been determined during linkage process. Perform until linkage process in the first pass and specify this assembler option in the second pass. For more information on the extended instruction and optimization, refer to Sections 8.6 and 8.7.

Default: The assembler does not optimizes extended instructions.

When entering options in the command line, you need to place one or more spaces before and after the option.

Example: **as -otest.o -adh1 test.s**

## 8.4 Scope

Symbols defined in each source file can freely be referred to within that file. Such reference range of symbols is termed scope.

Usually, reference can be made only within a defined file. If a symbol that does not exist in that file is referenced, the **as** assembler creates the object file assuming that the symbol is an undefined symbol, leaving the problem to be solved by the **ld** linker.

If your development project requires the use of multiple source files, it is necessary for the scope to be extended to cover other source files. The **as** assembler has the pseudo-instructions that can be used for this purpose.

Symbols that can be referenced in only the file where they are defined are called "local symbols". Symbols that are declared to be global are called "global symbols". Local symbols – even when symbols of the same name are specified in two or more different files – are handled as different symbols. Global symbols – if defined as overlapping in multiple files – cause a warning to be generated in the **ld** linker.

Example:

**file1:** file in which global symbol is defined

```
.global SYMBOL      ...Global declaration of symbols that are to be defined in this file.
.global VAR1

SYMBOL:
    :
    :

LABEL:               ...Local symbol
                    (Can be referred to only in this file)
    :
    .section .bss
    .align 2

VAR1:
    .zero 4
```

**file2:** file in which a global symbol is referred

```
xcall SYMBOL        ...Symbol externally referred
    :
xld.a %r1,VAR1      ...Symbol externally referred
LABEL:             ...Local symbol
                  (Treated as a different symbol from LABEL of file1)
    :
```

The **as** assembler regards the symbols `SYMBOL` and `VAR1` in the `file2` as those of undefined addresses in the assembling, and includes that information in the object file it delivers. Those addresses are finally determined by the processing of the **ld** linker.

## 8.5 Assembler Directives

---

The assembler directives are not converted to execution codes, but they are designed to control the assembler or to set data. For discrimination from other instructions, all the assembler directives begin with a period (.). Describe the directives in lowercase unless otherwise specified. Parameters are discriminated between uppercase and lowercase. The **as** assembler supports all the gnu assembler directives. Refer to the gnu assembler manual for details of the assembler directives. The following explains the often-utilized directives.

### 8.5.1 Text Section Defining Directive (`.text`)

#### Instruction format

```
.text
```

#### Description

Declares the start of a `.text` section. Statements following this instruction are assembled as those to be mapped in the `.text` section, until another section is declared.

## 8.5.2 Data Section Defining Directives (`.rodata`, `.data`)

### List of data section defining directives

- `.rodata`     Declares a `.rodata` section in which constants are located.
- `.data`       Declares a `.data` section in which data with initial values are located.

### Instruction format

```
.section .rodata
.section .data
```

### Description

#### (1) `.section .rodata`

Declares the start of a constant data section. Statements following this instruction are assembled as those to be mapped in the `.rodata` section, until another section is declared. Usually, this section will be mapped into a read-only memory at the stage of linkage.

Example: `.section .rodata`     Defines a `.rodata` section.

#### (2) `.section .data`

Declares the start of a data section with an initial value. Statements following this instruction are assembled as those to be mapped in the `.data` section, until another section is declared. Usually, this section will be mapped into a read-only memory at the stage of linkage and data in this section must be copied to a read/write memory such as a RAM by the software before using.

Example: `.section .data`       Defines a `.data` section.

### Note

The data space allocated by the data-define directive is as follows:

- 1 byte: `.byte`
- 2 bytes: `.short`, `.hword`, `.word`, `.int`
- 4 bytes: `.long`



### 8.5.3 Bss Section Defining Directive (`.bss`)

#### List of `bss` section defining directives

`.bss` Declares a `.bss` section for data without an initial value.

#### Instruction format

```
.section .bss
```

#### Description

Declares the start of a uninitialized data section. Statements following this instruction are assembled as those to be mapped in the `.bss` section, until another section is declared.

Example: `.section .bss` Defines a `.bss` section.

#### Note

- The labels described in the `.bss` section will be defined as local symbols by default. To define a global symbol, use the `.global` directive.

```
Example: .section .bss
        .align 2
VAR1:
        .skip 4           Defines the 4-byte local variable VAR1.
        .section .bss
        .global VAR2
        .align 2
VAR2:
        .skip 4           Defines the 4-byte global variable VAR2.
```

- Areas in `.bss` sections can be secured using the `.skip` directive. The `.space` directive cannot be used because it has an initial data.

## 8.5.4 Data Defining Directives (`.long`, `.short`, `.byte`, `.ascii`, `.space`)

The following assembler directives are used to define data in `.data` or `.text` sections:

### List of data defining directives

- `.long` Define 4-byte data.
- `.short` Define 2-byte data.
- `.byte` Define 1-byte data.
- `.ascii` Define ASCII character strings.
- `.space` Fills an area with a byte data.

### Instruction format

```
.long <4-byte data> [ , <4-byte data> ... , <4-byte data> ]
.short <2-byte data> [ , <2-byte data> ... , <2-byte data> ]
.byte <1-byte data> [ , <1-byte data> ... , <1-byte data> ]
.ascii "<character string>" [ , "<character string>" ... , "<character string>" ]
.space <length> [ , <1-byte data> ]
```

```
<4-byte data>      0x0-0xffffffff
<2-byte data>      0x0-0xffff
<1-byte data>      0x0-0xff
<character string> ASCII character string
<length>           Area size to be filled
```

### Description

#### (1) `.long`, `.short`, `.byte`

Defines one or more 4-byte data, 2-byte data, or 1-byte data. When specifying two or more data, separate them with a comma.

The defined data is located beginning with a boundary address matched to the data size by the data defining directive unless it is immediately preceded by the `.align` directive. If the current position is not a boundary address, 0x00 is set in the interval from that position to the nearest boundary address.

```
Example: .long  0x0, 0x1, 0x2
        .byte  0xff
```

In addition to these directives, the directives listed below can also be used.

```
.hword  same as .short
.word   same as .short
.int    same as .short
```

#### (2) `.ascii`

Defines one or more string literals. Enclose a character string in double quotes. ASCII characters and an escape sequence that begins with a symbol `"\"` can be written in a character string. For example, if you want to set double quote in a character string, write `\"`; to set a `\`, write `\\`.

When specifying two or more strings, separate them with a comma.

The defined data is located beginning with the current address first, unless it is immediately preceded by the `.align` directive.

```
Example: .ascii "abc", "xyz"
        .ascii "abc\"D\"efg"      (= abc"D"efg)
```

#### (3) `.space`

An area of the specified `<length>` bytes long is set to `<1-byte data>`. The area begins from the current address unless it is immediately preceded by the `.align` directive.

If `<1-byte data>` is omitted, the area is filled with 0x0. To fill the area with 0x0, the `.zero` directive (see the next page) can also be used.

```
Example: .space 4, 0xff      Sets 0xff to the 4-byte area beginning from the current address.
        .zero 4              (= .space 4, 0x0)
```

## 8.5.5 Area Securing Directive (`.zero`)

### Instruction format

```
.zero <length>
```

`<length>` Area size in bytes

### Description

This directive secures a `<length>` bytes of blank area in the current `.bss` section.

The area begins from the current address unless it is immediately preceded by the `.align` directive.

Example:

```
.section .bss
.global VAR1
.align 2
VAR1:
.zero 4           Secures an space for the 4-byte global variable VAR1.
```

## 8.5.6 Alignment Directive (`.align`)

### Instruction format

`.align <alignment>`

`<alignment>` Value to specify a boundary

### Description

The data that appears immediately after this directive is aligned to a  $2^n$  byte boundary ( $n = \text{<alignment>}$ ).

Example: `.align 2` Aligns the following data to a 4-byte boundary.

### Note

The `.align` directive is valid for only the immediately following data definition or area securing directive. Therefore, when defining data that requires alignment, you need to use the `.align` directive for each data definition directive.

## 8.5.7 Global Declaring Directive (`.global`)

### Instruction format

```
.global <symbol>
```

`<symbol>` Symbol to be defined in the current file

### Description

Makes global declaration of a symbol. The declaration made in a file with a symbol defined converts that symbol to a global symbol which can be referred to from other modules.

Example: `.global SUB1`

### Note

The symbols are always defined as a local symbol unless it is declared using this directive.

## 8.5.8 Symbol Defining Directive (`.set`)

### Instruction format

```
.set <symbol>, <address>
```

`<symbol>` Symbol for memory access (address reference)

`<address>` Absolute address

### Description

Defines a symbol with an absolute address (24-bit).

Example: `.set DATA1, 0x80000` Defines the symbol `DATA1` that represents absolute address `0x80000`.

### Note

The symbol is defined as a local symbol. To use it as a global symbol, global declaration using the `.global` directive is necessary.

## 8.6 Extended Instructions

The **as** assembler supports the extended instructions explained below. Extended instructions allow an operation that normally requires using multiple instructions including the **ext** instruction to be written in one instruction. They are expanded into the absolutely necessary minimum basic instructions according to instruction functionality and the operand's immediate size before assembling.

Symbols used in explanation

<i>immX</i>	Unsigned X-bit immediate
<i>signX</i>	Signed X-bit immediate
<i>symbol</i>	Symbol to indicate memory address
<i>label</i>	Jump address label
<i>(X:Y)</i>	Bit field from bit X to bit Y

### 8.6.1 Arithmetic Operation Instructions

#### Types and functions of extended instructions

Extended instruction	Function	Expansion
<b>sadd</b> %rd, imm16	$\%rd \leftarrow \%rd + imm16$	(1)
<b>sadc</b> %rd, imm16	$\%rd \leftarrow \%rd + imm16 + C$	(1)
<b>sadd.a</b> %rd, imm20	$\%rd \leftarrow \%rd + imm20$	(2)
<b>sadd.a</b> %sp, imm20	$\%sp \leftarrow \%sp + imm20$	(2)
<b>ssub</b> %rd, imm16	$\%rd \leftarrow \%rd - imm16$	(1)
<b>ssbc</b> %rd, imm16	$\%rd \leftarrow \%rd - imm16 - C$	(1)
<b>ssub.a</b> %rd, imm20	$\%rd \leftarrow \%rd - imm20$	(2)
<b>ssub.a</b> %sp, imm20	$\%sp \leftarrow \%sp - imm20$	(2)
<b>xadd</b> %rd, imm16	$\%rd \leftarrow \%rd + imm16$	(1)
<b>xadc</b> %rd, imm16	$\%rd \leftarrow \%rd + imm16 + C$	(1)
<b>xadd.a</b> %rd, imm24	$\%rd \leftarrow \%rd + imm24$	(3)
<b>xadd.a</b> %sp, imm24	$\%sp \leftarrow \%sp + imm24$	(3)
<b>xsub</b> %rd, imm16	$\%rd \leftarrow \%rd - imm16$	(1)
<b>xsbc</b> %rd, imm16	$\%rd \leftarrow \%rd - imm16 - C$	(1)
<b>xsub.a</b> %rd, imm24	$\%rd \leftarrow \%rd - imm24$	(3)
<b>xsub.a</b> %sp, imm24	$\%sp \leftarrow \%sp - imm24$	(3)

These extended instructions allow a 16-bit/20-bit/24-bit immediate to be specified directly in an add or subtract operation.

A conditional operation option (*/c*, */nc*) cannot be specified in the extended instructions.

#### Basic instructions after expansion

<b>sadd, xadd</b>	Expanded into the add instruction
<b>sadc, xadc</b>	Expanded into the adc instruction
<b>sadd.a, xadd.a</b>	Expanded into the add.a instruction
<b>ssub, xsub</b>	Expanded into the sub instruction
<b>ssbc, xsbc</b>	Expanded into the sbc instruction
<b>ssub.a, xsub.a</b>	Expanded into the sub.a instruction

## Expansion formats

(1) **sOP %rd,imm16/xOP %rd,imm16** (OP = add, adc, sub, sbc)

Example: `xadd %rd,imm16`

$imm16 \leq 0x7f$	$0x7f < imm16$
<code>add %rd,imm16(6:0)</code>	<code>ext imm16(15:7)</code> <code>add %rd,imm16(6:0)</code>

(2) **sOP.a %rd,imm20/sOP.a %sp,imm20** (OP = add, sub)

Example: `sadd.a %rd,imm20`

$imm20 \leq 0x7f$	$0x7f < imm20$
<code>add.a %rd,imm20(6:0)</code>	<code>ext imm20(19:7)</code> <code>add.a %rd,imm20(6:0)</code>

(3) **xOP.a %rd,imm24/xOP.a %sp,imm24** (OP = add, sub)

Example: `xadd.a %rd,imm24`

$imm24 \leq 0x7f$	$0x7f < imm24 \leq 0xffff$	$0xffff < imm24$
<code>add.a %rd,imm24(6:0)</code>	<code>ext imm24(19:7)</code> <code>add.a %rd,imm24(6:0)</code>	<code>ext imm24(23:20)</code> <code>ext imm24(19:7)</code> <code>add.a %rd,imm24(6:0)</code>



## 8.6.2 Comparison Instructions

### Types and functions of extended instructions

Extended instruction	Function	Expansion
<code>scmp %rd, imm16</code>	<code>%rd-imm16</code> (Sets/resets C, V, Z and N flags in PSR)	(1)
<code>scmc %rd, imm16</code>	<code>%rd-imm16-C</code> (Sets/resets C, V, Z and N flags in PSR)	(1)
<code>scmp.a %rd, imm20</code>	<code>%rd-imm20</code> (Sets/resets C, V, Z and N flags in PSR)	(2)
<code>xcmp %rd, imm16</code>	<code>%rd-imm16</code> (Sets/resets C, V, Z and N flags in PSR)	(1)
<code>xcmc %rd, imm16</code>	<code>%rd-imm16-C</code> (Sets/resets C, V, Z and N flags in PSR)	(1)
<code>xcmp.a %rd, imm24</code>	<code>%rd-imm24</code> (Sets/resets C, V, Z and N flags in PSR)	(3)

These extended instructions allow you to compare a general-purpose register and a 16-bit/20-bit/24-bit immediate.

A conditional operation option (`/c`, `/nc`) cannot be specified in the extended instructions.

### Basic instructions after expansion

`scmp`, `xcmp` Expanded into the `cmp` instruction  
`scmc`, `xcmc` Expanded into the `cmc` instruction  
`scmp.a`, `xcmp.a` Expanded into the `cmp.a` instruction

### Expansion formats

(1) `sOP %rd, imm16 / xOP %rd, imm16` ( $OP = \text{cmp, cmc}$ )

Example: `xcmp %rd, imm16`

$imm16 \leq 0x7f$	$0x7f < imm16$
<code>cmp %rd, imm16(6:0)</code>	<code>ext imm16(15:7)</code> <code>cmp %rd, imm16(6:0)</code>

(2) `scmp.a %rd, imm20`

$imm20 \leq 0x7f$	$0x7f < imm20$
<code>cmp.a %rd, imm20(6:0)</code>	<code>ext imm20(19:7)</code> <code>cmp.a %rd, imm20(6:0)</code>

(3) `xcmp.a %rd, imm24`

$imm24 \leq 0x7f$	$0x7f < imm24 \leq 0xffff$	$0xffff < imm24$
<code>cmp.a %rd, imm24(6:0)</code>	<code>ext imm24(19:7)</code> <code>cmp.a %rd, imm24(6:0)</code>	<code>ext imm24(23:20)</code> <code>ext imm24(19:7)</code> <code>cmp.a %rd, imm24(6:0)</code>

## 8.6.3 Logic Operation Instructions

### Types and functions of extended instructions

Extended instruction	Function	Expansion
<code>sand %rd, imm16</code>	$\%rd \leftarrow \%rd \& imm16$	(1)
<code>soor %rd, imm16</code>	$\%rd \leftarrow \%rd   imm16$	(1)
<code>sxor %rd, imm16</code>	$\%rd \leftarrow \%rd \wedge imm16$	(1)
<code>snot %rd, imm16</code>	$\%rd \leftarrow !imm16$	(1)
<code>xand %rd, imm16</code>	$\%rd \leftarrow \%rd \& imm16$	(1)
<code>xoor %rd, imm16</code>	$\%rd \leftarrow \%rd   imm16$	(1)
<code>xxor %rd, imm16</code>	$\%rd \leftarrow \%rd \wedge imm16$	(1)
<code>snot %rd, imm16</code>	$\%rd \leftarrow !imm16$	(1)

These extended instructions allow a 16-bit immediate to be specified directly in a logical operation. A conditional operation option (*/c*, */nc*) cannot be specified in the extended instructions.

### Basic instructions after expansion

**sand, xand** Expanded into the `and` instruction  
**soor, xoor** Expanded into the `or` instruction  
**sxor, xxor** Expanded into the `xor` instruction  
**snot, xnot** Expanded into the `not` instruction

### Expansion format

(1) `sOP %rd, imm16 / xOP %rd, imm16` (*OP* = `and`, `oor`, `xor`, `not`)

Example: `xand %rd, imm16`

$imm16 \leq 0x7f$	$0x7f < imm16$
<code>and %rd, imm16(6:0)</code>	<code>ext imm16(15:7)</code> <code>and %rd, imm16(6:0)</code>

## 8.6.4 Data Transfer Instructions (between Stack and Register)

### Types and functions of extended instructions

Extended instruction	Function	Expansion
<code>sld.b %rd, [%sp+imm20]</code>	$\%rd \leftarrow B[\%sp+imm20]$ (with sign extension)	(1)
<code>sld.ub %rd, [%sp+imm20]</code>	$\%rd \leftarrow B[\%sp+imm20]$ (with zero extension)	(1)
<code>sld %rd, [%sp+imm20]</code>	$\%rd \leftarrow W[\%sp+imm20]$	(1)
<code>sld.a %rd, [%sp+imm20]</code>	$\%rd \leftarrow A[\%sp+imm20]$ (23:0), ignored $\leftarrow A[\%sp+imm20]$ (31:24)	(1)
<code>sld.b [%sp+imm20], %rs</code>	$B[\%sp+imm20] \leftarrow \%rs(7:0)$	(1)
<code>sld [%sp+imm20], %rs</code>	$W[\%sp+imm20] \leftarrow \%rs(15:0)$	(1)
<code>sld.a [%sp+imm20], %rs</code>	$A[\%sp+imm20]$ (23:0) $\leftarrow \%rs(23:0)$ , $A[\%sp+imm20]$ (31:24) $\leftarrow 0$	(1)
<code>xld.b %rd, [%sp+imm24]</code>	$\%rd \leftarrow B[\%sp+imm24]$ (with sign extension)	(2)
<code>xld.ub %rd, [%sp+imm24]</code>	$\%rd \leftarrow B[\%sp+imm24]$ (with zero extension)	(2)
<code>xld %rd, [%sp+imm24]</code>	$\%rd \leftarrow W[\%sp+imm24]$	(2)
<code>xld.a %rd, [%sp+imm24]</code>	$\%rd \leftarrow A[\%sp+imm24]$ (23:0), ignored $\leftarrow A[\%sp+imm24]$ (31:24)	(2)
<code>xld.b [%sp+imm24], %rs</code>	$B[\%sp+imm24] \leftarrow \%rs(7:0)$	(2)
<code>xld [%sp+imm24], %rs</code>	$W[\%sp+imm24] \leftarrow \%rs(15:0)$	(2)
<code>xld.a [%sp+imm24], %rs</code>	$A[\%sp+imm24]$ (23:0) $\leftarrow \%rs(23:0)$ , $A[\%sp+imm24]$ (31:24) $\leftarrow 0$	(2)

These extended instructions allow you to directly specify a displacement of up to 20 bits/24 bits. Specification of *imm20/imm24* can be omitted.

### Basic instructions after expansion

- `sld.b, xld.b` Expanded into the `ld.b` instruction
- `sld.ub, xld.ub` Expanded into the `ld.ub` instruction
- `sld, xld` Expanded into the `ld` instruction
- `sld.a, xld.a` Expanded into the `ld.a` instruction

### Expansion formats

If *imm20/imm24* is omitted, the `as` assembler assumes that `[%sp+0x0]` is specified as it expands the instruction.

- (1) `sOP %rd, [%sp+imm20]` (*OP* = `ld.b`, `ld.ub`, `ld`, `ld.a`)  
`sOP [%sp+imm20], %rs` (*OP* = `ld.b`, `ld`, `ld.a`)

Example: `sld.a %rd, [%sp+imm20]`

$imm20 \leq 0x7f$	$0x7f < imm20$
<code>ld.a %rd, [%sp+imm20(6:0)]</code>	<code>ext imm20(19:7)</code> <code>ld.a %rd, [%sp+imm20(6:0)]</code>

- (2) `xOP %rd, [%sp+imm24]` (*OP* = `ld.b`, `ld.ub`, `ld`, `ld.a`)  
`xOP [%sp+imm24], %rs` (*OP* = `ld.b`, `ld`, `ld.a`)

Example: `xld.a %rd, [%sp+imm24]`

$imm24 \leq 0x7f$	$0x7f < imm24 \leq 0xffff$	$0xffff < imm24$
<code>ld.a %rd, [%sp+imm24(6:0)]</code>	<code>ext imm24(19:7)</code> <code>ld.a %rd, [%sp+imm24(6:0)]</code>	<code>ext imm24(23:20)</code> <code>ext imm24(19:7)</code> <code>ld.a %rd, [%sp+imm24(6:0)]</code>

## 8.6.5 Data Transfer Instructions (between Memory and Register)

### Types and functions of extended instructions

Extended instruction	Function	Expansion
<code>sld.b %rd, [imm20]</code>	$\%rd \leftarrow B[imm20]$ (with sign extension)	(1)
<code>sld.ub %rd, [imm20]</code>	$\%rd \leftarrow B[imm20]$ (with zero extension)	(1)
<code>sld %rd, [imm20]</code>	$\%rd \leftarrow W[imm20]$	(1)
<code>sld.a %rd, [imm20]</code>	$\%rd \leftarrow A[imm20] (23:0)$ , ignored $\leftarrow A[imm20] (31:24)$	(1)
<code>sld.b [imm20], %rs</code>	$B[imm20] \leftarrow \%rs (7:0)$	(1)
<code>sld [imm20], %rs</code>	$W[imm20] \leftarrow \%rs (15:0)$	(1)
<code>sld.a [imm20], %rs</code>	$A[imm20] (23:0) \leftarrow \%rs (23:0)$ , $A[imm20] (31:24) \leftarrow 0$	(1)
<code>xld.b %rd, [imm24]</code>	$\%rd \leftarrow B[imm24]$ (with sign extension)	(2)
<code>xld.ub %rd, [imm24]</code>	$\%rd \leftarrow B[imm24]$ (with zero extension)	(2)
<code>xld %rd, [imm24]</code>	$\%rd \leftarrow W[imm24]$	(2)
<code>xld.a %rd, [imm24]</code>	$\%rd \leftarrow A[imm24] (23:0)$ , ignored $\leftarrow A[imm24] (31:24)$	(2)
<code>xld.b [imm24], %rs</code>	$B[imm24] \leftarrow \%rs (7:0)$	(2)
<code>xld [imm24], %rs</code>	$W[imm24] \leftarrow \%rs (15:0)$	(2)
<code>xld.a [imm24], %rs</code>	$A[imm24] (23:0) \leftarrow \%rs (23:0)$ , $A[imm24] (31:24) \leftarrow 0$	(2)

These extended instructions allow memory locations to be accessed by specifying the address with a 20-bit/24-bit immediate. However, the postincrement function (`[] +`) cannot be used.

### Basic instructions after expansion

<code>sld.b, xld.b</code>	Expanded into the <code>ld.b</code> instruction
<code>sld.ub, xld.ub</code>	Expanded into the <code>ld.ub</code> instruction
<code>sld, xld</code>	Expanded into the <code>ld</code> instruction
<code>sld.a, xld.a</code>	Expanded into the <code>ld.a</code> instruction

### Expansion formats

- (1) `sOP %rd, [imm20]` ( $OP = ld.b, ld.ub, ld, ld.a$ )  
`sOP [imm20], %rs` ( $OP = ld.b, ld, ld.a$ )

Example: `sld.a %rd, [imm20]`

$imm20 \leq 0x7f$	$0x7f < imm20$
<code>ld.a %rd, [imm20(6:0)]</code>	ext $imm20(19:7)$ <code>ld.a %rd, [imm20(6:0)]</code>

- (2) `xOP %rd, [imm24]` ( $OP = ld.b, ld.ub, ld, ld.a$ )  
`xOP [imm24], %rs` ( $OP = ld.b, ld, ld.a$ )

Example: `xld.a %rd, [imm24]`

$imm24 \leq 0x7f$	$0x7f < imm24 \leq 0xffff$	$0xffff < imm24$
<code>ld.a %rd, [imm24(6:0)]</code>	ext $imm24(19:7)$ <code>ld.a %rd, [imm24(6:0)]</code>	ext $imm24(23:20)$ ext $imm24(19:7)$ <code>ld.a %rd, [imm24(6:0)]</code>

## 8.6.6 Immediate Data Load Instructions

### Types and functions of extended instructions

Extended instruction	Function	Expansion
<code>sld %rd, imm16</code>	$\%rd \leftarrow imm16$	(1)
<code>sld.a %rd, imm20</code>	$\%rd \leftarrow imm20$	(2)
<code>sld.a %sp, imm20</code>	$\%sp \leftarrow imm20$	(2)
<code>sld %rd, symbol+imm16</code>	$\%rd \leftarrow symbol+imm16(15:0)$	(4)
<code>sld.a %rd, symbol+imm20</code>	$\%rd \leftarrow symbol+imm20(19:0)$	(5)
<code>sld.a %sp, symbol+imm20</code>	$\%sp \leftarrow symbol+imm20(19:0)$	(5)
<code>xld %rd, imm16</code>	$\%rd \leftarrow imm16$	(1)
<code>xld.a %rd, imm24</code>	$\%rd \leftarrow imm24$	(3)
<code>xld.a %sp, imm24</code>	$\%sp \leftarrow imm24$	(3)
<code>xld %rd, symbol+imm16</code>	$\%rd \leftarrow symbol+imm16(15:0)$	(4)
<code>xld.a %rd, symbol+imm24</code>	$\%rd \leftarrow symbol+imm24(23:0)$	(6)
<code>xld.a %sp, symbol+imm24</code>	$\%sp \leftarrow symbol+imm24(23:0)$	(6)

These extended instructions allow a 16-bit/20-bit/24-bit immediate to be loaded directly into a general-purpose register. A symbol also can be used for immediate specification.

### Basic instructions after expansion

- `sld, xld` Expanded into the `ld` instruction
- `sld.a, xld.a` Expanded into the `ld.a` instruction

### Expansion formats

#### (1) `sld %rd, imm16 / xld %rd, imm16`

Example: `xld %rd, imm16`

$imm16 \leq 0x7f$	$0x7f < imm16$
<code>ld %rd, imm16(6:0)</code>	<code>ext imm16(15:7)</code> <code>ld %rd, imm16(6:0)</code>

#### (2) `sld.a %rd, imm20 / sld.a %sp, imm20`

Example: `sld.a %rd, imm20`

$imm20 \leq 0x7f$	$0x7f < imm20$
<code>ld.a %rd, imm20(6:0)</code>	<code>ext imm20(19:7)</code> <code>ld.a %rd, imm20(6:0)</code>

#### (3) `xld.a %rd, imm24 / xld.a %sp, imm24`

Example: `xld.a %rd, imm24`

$imm24 \leq 0x7f$	$0x7f < imm24 \leq 0xffff$	$0xffff < imm24$
<code>ld.a %rd, imm24(6:0)</code>	<code>ext imm24(19:7)</code> <code>ld.a %rd, imm24(6:0)</code>	<code>ext imm24(23:20)</code> <code>ext imm24(19:7)</code> <code>ld.a %rd, imm24(6:0)</code>

#### (4) `sld %rd, symbol+imm16 / xld %rd, symbol+imm16`

Example: `sld %rd, symbol+imm16`

Unconditional
<code>ext (symbol+imm16)(15:7)</code> <code>ld %rd, (symbol+imm16)(6:0)</code>

#### (5) `sld.a %rd, symbol+imm20 / sld.a %sp, symbol+imm20`

Example: `sld.a %rd, symbol+imm20`

Unconditional
<code>ext (symbol+imm20)(19:7)</code> <code>ld.a %rd, (symbol+imm20)(6:0)</code>

(6) `xld.a %rd, symbol±imm24 / xld.a %sp, symbol±imm24`

Example: `xld.a %rd, symbol±imm24`

Unconditional	
<code>ext</code>	<code>(symbol±imm24) (23:20)</code>
<code>ext</code>	<code>(symbol±imm24) (19:7)</code>
<code>ld.a</code>	<code>%rd, (symbol±imm24) (6:0)</code>

## 8.6.7 Branch Instructions

### Types and functions of extended instructions

Extended instruction	Function	Expansion
scall <i>label+imm20</i>	PC relative subroutine call	(1)
sjpr <i>label+imm20</i>	PC relative unconditional jump	(1)
sjreq <i>label+imm20</i>	PC relative conditional jump	(2)
sjrne <i>label+imm20</i>	PC relative conditional jump	(2)
sjrgt <i>label+imm20</i>	PC relative conditional jump	(2)
sjrge <i>label+imm20</i>	PC relative conditional jump	(2)
sjrlt <i>label+imm20</i>	PC relative conditional jump	(2)
sjrle <i>label+imm20</i>	PC relative conditional jump	(2)
sjrugt <i>label+imm20</i>	PC relative conditional jump	(2)
sjruge <i>label+imm20</i>	PC relative conditional jump	(2)
sjrult <i>label+imm20</i>	PC relative conditional jump	(2)
sjrule <i>label+imm20</i>	PC relative conditional jump	(2)
scalla <i>label+imm20</i>	PC absolute subroutine call	(3)
sjpa <i>label+imm20</i>	PC absolute unconditional jump	(3)
scall <i>sign20</i>	PC relative subroutine call	(4)
sjpr <i>sign20</i>	PC relative unconditional jump	(4)
sjreq <i>sign20</i>	PC relative conditional jump	(5)
sjrne <i>sign20</i>	PC relative conditional jump	(5)
sjrgt <i>sign20</i>	PC relative conditional jump	(5)
sjrge <i>sign20</i>	PC relative conditional jump	(5)
sjrlt <i>sign20</i>	PC relative conditional jump	(5)
sjrle <i>sign20</i>	PC relative conditional jump	(5)
sjrugt <i>sign20</i>	PC relative conditional jump	(5)
sjruge <i>sign20</i>	PC relative conditional jump	(5)
sjrult <i>sign20</i>	PC relative conditional jump	(5)
sjrule <i>sign20</i>	PC relative conditional jump	(5)
scalla <i>imm20</i>	PC absolute subroutine call	(6)
sjpa <i>imm20</i>	PC absolute unconditional jump	(6)
xcall <i>label+imm24</i>	PC relative subroutine call	(7)
xjpr <i>label+imm24</i>	PC relative unconditional jump	(7)
xjreq <i>label+imm24</i>	PC relative conditional jump	(8)
xjrne <i>label+imm24</i>	PC relative conditional jump	(8)
xjrgt <i>label+imm24</i>	PC relative conditional jump	(8)
xjrge <i>label+imm24</i>	PC relative conditional jump	(8)
xjrlt <i>label+imm24</i>	PC relative conditional jump	(8)
xjrle <i>label+imm24</i>	PC relative conditional jump	(8)
xjrugt <i>label+imm24</i>	PC relative conditional jump	(8)
xjruge <i>label+imm24</i>	PC relative conditional jump	(8)
xjrult <i>label+imm24</i>	PC relative conditional jump	(8)
xjrule <i>label+imm24</i>	PC relative conditional jump	(8)
xcalla <i>label+imm24</i>	PC absolute subroutine call	(9)
xjpa <i>label+imm24</i>	PC absolute unconditional jump	(9)
xcall <i>sign24</i>	PC relative subroutine call	(10)
xjpr <i>sign24</i>	PC relative unconditional jump	(10)
xjreq <i>sign24</i>	PC relative conditional jump	(11)
xjrne <i>sign24</i>	PC relative conditional jump	(11)
xjrgt <i>sign24</i>	PC relative conditional jump	(11)
xjrge <i>sign24</i>	PC relative conditional jump	(11)
xjrlt <i>sign24</i>	PC relative conditional jump	(11)
xjrle <i>sign24</i>	PC relative conditional jump	(11)
xjrugt <i>sign24</i>	PC relative conditional jump	(11)
xjruge <i>sign24</i>	PC relative conditional jump	(11)
xjrult <i>sign24</i>	PC relative conditional jump	(11)
xjrule <i>sign24</i>	PC relative conditional jump	(11)
xcalla <i>imm24</i>	PC absolute subroutine call	(12)
xjpa <i>imm24</i>	PC absolute unconditional jump	(12)

These extended instructions allow a branch destination to be specified using a 20-bit/24-bit immediate or a label. The branch conditions of these conditional jump instructions are the same as those of the basic instructions.

The extended instructions can be used as delayed branch instructions by adding ".d".

Example: `xcall.d sign24`

### Basic instructions after expansion

<code>scall, scall.d, xcall, xcall.d</code>	Expanded into the <code>call/call.d</code> instruction
<code>scalla, scalla.d, xcalla, xcalla.d</code>	Expanded into the <code>calla/calla.d</code> instruction
<code>sjpa, sjpa.d, xjpa, xjpa.d</code>	Expanded into the <code>jpa/jpa.d</code> instruction
<code>sjpr, sjpr.d, xjpr, xjpr.d</code>	Expanded into the <code>jpr/jpr.d</code> instruction
<code>sjreq, sjreq.d, xjreq, xjreq.d</code>	Expanded into the <code>jreq/jreq.d</code> instruction
<code>sjrne, sjrne.d, xjrne, xjrne.d</code>	Expanded into the <code>jrne/jrne.d</code> instruction
<code>sjrgt, sjrgt.d, xjrgt, xjrgt.d</code>	Expanded into the <code>jrgt/jrgt.d</code> instruction
<code>sjrge, sjrge.d, xjrge, xjrge.d</code>	Expanded into the <code>jrge/jrge.d</code> instruction
<code>sjrlt, sjrlt.d, xjrlt, xjrlt.d</code>	Expanded into the <code>jrlt/jrlt.d</code> instruction
<code>sjrle, sjrle.d, xjrle, xjrle.d</code>	Expanded into the <code>jrle/jrle.d</code> instruction
<code>sjrugt, sjrugt.d, xjrugt, xjrugt.d</code>	Expanded into the <code>jrugt/jrugt.d</code> instruction
<code>sjruge, sjruge.d, xjruge, xjruge.d</code>	Expanded into the <code>jruge/jruge.d</code> instruction
<code>sjrult, sjrult.d, xjrult, xjrult.d</code>	Expanded into the <code>jrult/jrult.d</code> instruction
<code>sjrule, sjrule.d, xjrle, xjrle.d</code>	Expanded into the <code>jrle/jrle.d</code> instruction

### Expansion formats

- (1) `sOP label+imm20` ( $OP = \text{call, call.d, jpr, jpr.d}$ )

Example: `scall label+imm20`

Unconditional	
<code>ext</code>	<code>(label+imm20) (19:12)</code>
<code>call</code>	<code>(label+imm20) (11:1)</code>

- (2) `sOP label+imm20` ( $OP = \text{jr*}, \text{jr*.d}$ )

Example: `sjreq label+imm20`

Unconditional	
<code>ext</code>	<code>(label+imm20) (19:8)</code>
<code>jreq</code>	<code>(label+imm20) (7:1)</code>

- (3) `sOP label+imm20` ( $OP = \text{calla, calla.d, jpa, jpa.d}$ )

Example: `scalla label+imm20`

Unconditional	
<code>ext</code>	<code>(label+imm20) (19:7)</code>
<code>calla</code>	<code>(label+imm20) (6:0)</code>

- (4) `sOP sign20` ( $OP = \text{call, call.d, jpr, jpr.d}$ )

Example: `scall sign20`

$-1024 \leq \text{sign20} \leq 1023$	$\text{sign20} < -1024$ or $1023 < \text{sign20}$
<code>call sign20(11:1)</code>	<code>ext sign20(19:12)</code> <code>call sign20(11:1)</code>

- (5) `sOP sign20` ( $OP = \text{jr*}, \text{jr*.d}$ )

Example: `sjreq sign20`

$-128 \leq \text{sign20} \leq 127$	$\text{sign20} < -128$ or $127 < \text{sign20}$
<code>jreq sign20(7:1)</code>	<code>ext sign20(19:8)</code> <code>jreq sign20(7:1)</code>



**(6) sOP imm20** (OP = calla, calla.d, jpa, jpa.d)

Example: scalla imm20

$imm20 \leq 0x7f$	$0x7f < imm20$
calla imm20(6:0)	ext imm20(19:7) calla imm20(6:0)

**(7) xOP label+imm24** (OP = call, call.d, jpr, jpr.d)

Example: xcall label+imm24

Unconditional
ext (label+imm24) (23:12) call (label+imm24) (11:1)

**(8) xOP label+imm24** (OP = jr\*, jr\*.d)

Example: xjreq label+imm24

Unconditional
ext (label+imm24) (23:21) ext (label+imm24) (20:8) jreq (label+imm24) (7:1)

**(9) xOP label+imm24** (OP = calla, calla.d, jpa, jpa.d)

Example: xcalla label+imm24

Unconditional
ext (label+imm24) (23:20) ext (label+imm24) (19:7) calla (label+imm24) (6:0)

**(10) xOP sign24** (OP = call, call.d, jpr, jpr.d)

Example: xcall sign24

$-1024 \leq sign24 \leq 1023$	$sign24 < -1024$ or $1023 < sign24$
call sign24(11:1)	ext sign24(23:12) call sign24(11:1)

**(11) xOP sign24** (OP = jr\*, jr\*.d)

Example: xjreq sign24

$-128 \leq sign24 \leq 127$	$-1048576 \leq sign24 < -128$ or $127 < sign24 \leq 1048575$	$sign24 < -1048576$ or $1048575 < sign24$
jreq sign24(7:1)	ext sign24(20:8) jreq sign24(7:1)	ext sign24(23:21) ext sign24(20:8) jreq sign24(7:1)

**(12) xOP imm24** (OP = calla, calla.d, jpa, jpa.d)

Example: xcalla imm24

$imm24 \leq 0x7f$	$0x7f < imm24 \leq 0xffff$	$0xffff < imm24$
calla imm24(6:0)	ext imm24(19:7) calla imm24(6:0)	ext imm24(23:20) ext imm24(19:7) calla imm24(6:0)

## 8.6.8 Coprocessor Instructions

### Types and functions of extended instructions

Extended instruction	Function	Expansion
<code>sld.cw %rd, imm20</code>	Coprocessor ← %rd & imm20	(1)
<code>sld.ca %rd, imm20</code>	Coprocessor ← %rd & imm20, get results and flag statuses	(1)
<code>sld.cf %rd, imm20</code>	Coprocessor ← %rd & imm20, get flag statuses	(1)
<code>sld.cw %rd, symbol+imm20</code>	Coprocessor ← %rd & symbol+imm20	(2)
<code>sld.ca %rd, symbol+imm20</code>	Coprocessor ← %rd & symbol+imm20, get results and flag statuses	(2)
<code>sld.cf %sp, symbol+imm20</code>	Coprocessor ← %rd & symbol+imm20, get flag statuses	(2)
<code>xld.cw %rd, imm24</code>	Coprocessor ← %rd & imm24	(3)
<code>xld.ca %rd, imm24</code>	Coprocessor ← %rd & imm24, get results and flag statuses	(3)
<code>xld.cf %rd, imm24</code>	Coprocessor ← %rd & imm24, get flag statuses	(3)
<code>xld.cw %rd, symbol+imm24</code>	Coprocessor ← %rd & symbol+imm24	(4)
<code>xld.ca %rd, symbol+imm24</code>	Coprocessor ← %rd & symbol+imm24, get results and flag statuses	(4)
<code>xld.cf %rd, symbol+imm24</code>	Coprocessor ← %rd & symbol+imm24, get flag statuses	(4)

These extended instructions allow a 20-bit/24-bit immediate to be transferred to the coprocessor. A symbol also can be used for immediate specification.

### Basic instructions after expansion

`sld.cw, xld.cw` Expanded into the `ld.cw` instruction  
`sld.ca, xld.ca` Expanded into the `ld.ca` instruction  
`sld.cf, xld.cf` Expanded into the `ld.cf` instruction

### Expansion formats

(1) `sOP %rd, imm20` ( $OP = ld.cw, ld.ca, ld.cf$ )

Example: `sld.ca %rd, imm20`

$imm20 \leq 0x7f$	$0x7f < imm20$
<code>ld.ca %rd, imm20(6:0)</code>	<code>ext imm20(19:7)</code> <code>ld.ca %rd, imm20(6:0)</code>

(2) `sOP %rd, symbol+imm20` ( $OP = ld.cw, ld.ca, ld.cf$ )

Example: `sld.ca %rd, symbol+imm20`

Unconditional
<code>ext (symbol+imm20)(19:7)</code> <code>ld.ca %rd, (symbol+imm20)(6:0)</code>

(3) `xOP %rd, imm24` ( $OP = ld.cw, ld.ca, ld.cf$ )

Example: `xld.ca %rd, imm24`

$imm24 \leq 0x7f$	$0x7f < imm24 \leq 0xffff$	$0xffff < imm24$
<code>ld.ca %rd, imm24(6:0)</code>	<code>ext imm24(19:7)</code> <code>ld.ca %rd, imm24(6:0)</code>	<code>ext imm24(23:20)</code> <code>ext imm24(19:7)</code> <code>ld.ca %rd, imm24(6:0)</code>

(4) `xOP %rd, symbol+imm24` ( $OP = ld.cw, ld.ca, ld.cf$ )

Example: `xld.ca %rd, symbol+imm24`

Unconditional
<code>ext (symbol+imm24)(23:20)</code> <code>ext (symbol+imm24)(19:7)</code> <code>ld.ca %rd, (symbol+imm24)(6:0)</code>

## 8.7 Optimization of Extended Instructions

The C compiler compiles all codes that reference a global symbol address and some codes that reference a local symbol address into `s*` or `x*` extended instructions. As shown in the preceding section, these extended instructions are expanded into a basic instruction with the `ext` instructions.

If the operand of an extended instruction is an immediate data, the assembler adds zero to two `ext` instructions to the basic instruction according to the immediate data size. So the optimization is completed at this point.

If the operand of an extended instruction is a symbol, the assembler adds the required number of `ext` instructions for referencing to the entire memory space, as the symbol value is not determined until linkage has completed. Therefore, unnecessary `ext` instructions may be output and it increases the code size. The `-mc17_ext` option is used for the optimization to remove unnecessary `ext` instructions.

### `-mc17_ext` option

```
-mc17_ext filename.dump filename.map
```

`filename.dump` File in which global and local symbol information is described. Use **objdump** to create this file. The file name extension must be ".dump".

```
objdump -t filename.elf > filename.dump
```

`filename.map` File in which object mapping information is described. This file is delivered by the **ld** linker. The file name extension must be ".map".

```
Example: objdump -t test.elf > test.dump
         as -mc17_ext test.dump test.map -o test.o test.s
```

### Optimize procedure

The assembler needs the symbol address information determined in the linkage process for the optimization and gets it from the link map and dump files specified with the `-mc17_ext` option. Therefore, the optimization needs a two-pass make process. The procedure shown below should be written in the makefile to perform a two-pass make process.

1. Compile
2. Assemble (without the `-mc17_ext` option)
3. Link (with a map file output)
4. Create a dump file from the elf file using **objdump**.
5. Reassemble (with the `-mc17_ext` option) \* Specify the same input/output files as Step 2.

The makefile created by the **IDE** contains the procedure above.

## Optimize patterns

The optimization process is classified under five patterns.

### Optimization 1: for data transfer instructions

Example:

```
sub:
    ret

main:
    xsub.a    %sp, 0x4
    :
    xld.b    %r0, [sub]
    add.a    %sp, 0x4
    ret
```

The assembler obtains the address of the symbol (`sub`) from the dump and map files to determine the number of `ext` instructions to be used.

Symbol address = 0 to 0x7f: The extended instruction (`xld.b`) is expanded with no `ext` used.

Symbol address = 0x80 to 0xffff: The extended instruction (`xld.b`) is expanded with one `ext` used.

Symbol address = 0xffff to 0xfffff: The extended instruction (`xld.b`) is expanded with two `ext` used.

### Optimization 2: for arithmetic operation instructions

Example:

```
sub:
    ret

main:
    xsub.a    %sp, 0x4
    :
    xadd     %r1, sub
    add.a    %sp, 0x4
    ret
```

The assembler obtains the address of the symbol (`sub`) from the dump and map files to determine the number of `ext` instructions to be used.

Symbol address = 0 to 0x7f: The extended instruction (`xadd`) is expanded with no `ext` used.

Symbol address = 0x80 to 0xffff: The extended instruction (`xadd`) is expanded with one `ext` used.

### Optimization 3: for relative branch instructions

(`scall/xcall/sjpr/xjpr/sjrx/xjrx` instructions)

Example:

```
sub:
    ret

main:
    xsub.a    %sp, 0x4
    :
    xcall     sub          ← ADDR1
    add.a    %sp, 0x4
    ret
```

The assembler obtains the destination address (`sub`) directly from the dump and map files. The branch address (`ADDR1`) is not included in the dump and map files. Therefore, the assembler counts the instructions from the beginning address of the `main` routine that can be obtained from the dump and map files to calculate the distance from `main` to `ADDR1`. The number of `ext` instructions is determined from the branch distance calculated by subtracting the branch address from the destination address.

`scall/xcall/sjpr/xjpr` instructions

Branch distance = -1024 to 1022:

The extended instruction is expanded with no `ext` used.

Branch distance = -16,777,216 to -1026 or 1024 to 16,777,214:

The extended instruction is expanded with one `ext` used.

**sjrxx/xjrxx instructions**

Branch distance = -128 to 126:

The extended instruction is expanded with no `ext` used.

Branch distance = -1,048,576 to -130 or 128 to 1,048,574:

The extended instruction is expanded with one `ext` used.

Branch distance = -16,777,216 to -1,048,578 or 1,048,576 to 16,777,214:

The extended instruction is expanded with two `ext` used.

Note that `ext` instructions may be removed even in a case other than "`ext 0`". In the example below, no `ext` instruction is required since operand 3 will be automatically sign extended if all the bits of operands 1 and 2 and the MSB of operand 3 are 1. In this case, the added `ext` instructions are removed.

Example:

```
L1:
    nop
    xjrgt L1 ; (L1 - branch address - 2) >> 1 → -2
```

Before optimization	After optimization
<code>nop</code>	<code>nop</code>
<code>ext 0x7</code> ← operand 1 ( <i>imm3</i> )	<code>jrgt 0x7e</code>
<code>ext 0x1fff</code> ← operand 2 ( <i>imm13</i> )	
<code>jrgt 0x7c</code> ← operand 3 ( <i>sign7</i> )	

**Optimization 4: for absolute branch instructions (`scalla/xcalla/sjpa/xjpa` instructions)**

Example:

```
sub:
    ret

main:
    xsub.a %sp, 0x4
    :
    xcalla sub
    add.a %sp, 0x4
    ret
```

The assembler obtains the destination address (`sub`) from the dump and map files and determines the number of `ext` instructions to be used directly from the destination address.

Destination address = 0 to 0x7f: The extended instruction is expanded with no `ext` used.

Destination address = 0x80 to 0xffff: The extended instruction is expanded with one `ext` used.

Destination address = 0xffff to 0xfffff: The extended instruction is expanded with two `ext` used.

**Optimization 5: when the operand is an expression using a symbol**

Example:

```
sub:
    ret

main:
    xsub.a %sp, 0x4
    :
    xldb %r0, [sub+0x2]
    add.a %sp, 0x4
    ret
```

The assembler evaluates the expression in the operand and determines the number of `ext` instructions to be used.

Evaluation result = 0 to 0x7f: The extended instruction is expanded with no `ext` used.

Evaluation result = 0x80 to 0xffff: The extended instruction is expanded with one `ext` used.

Evaluation result = 0xffff to 0xfffff: The extended instruction is expanded with two `ext` used.

## Notes

- When the `ext` instructions are written directly to an assembler source to extend basic instructions, they are not optimized. When creating an assembler source, use `s*` or `x*` extended instructions for data transfer, arithmetic operation, and branch if a symbol is used as the operand.
- If the branch destination symbol for a relative branch instruction is located in another section, the branch instruction will not be optimized.

```
Example: .text_A 0x00000000 :
{
    __START__ text_A = . ;
    sub_1.o(.text)
    sub_2.o(.text)
    sub_3.o(.text)
    sub_4.o(.text)
    __END__ text_A = . ;
}

.text_B 0x00C00000 :
{
    __START__ text_B = . ;
    main.o(.text)
    __END__ text_B = . ;
}
```

In this example, the extended instructions for referencing or branching to a symbol between "main.o" and "sub\_X.o" will not be optimized. Therefore, new sections should not be added in the linker script file (.lds) if they are not necessary.

- When a file name is described twice or more in a dump file (source files with the same name (except extension and path) exist), optimization for local symbols using the dump file will not be performed. This is not applied when the uppercase and lowercase letter configuration is different even if the file name is the same. Furthermore, this condition does not affect optimization for global symbols using a map file.

## 8.8 Error/Warning Messages

The following shows the principal error and warning messages output by the assembler as:

Table 8.8.1 Error messages

Error message	Description
Error: Unrecognized opcode: 'XXXXX'	The operation code XXXXX is undefined.
Error: junk at end of line: 'XXXXX'	A format error of the operand.
Error: XXXXXX: invalid register name	The specified register cannot be used.
Error: operand out of range (XXXXXX: XXX not between AAA and BBB)	The value specified in the operand is out of the effective range.
Error: There are too many characters of one line in assembler source file. *	The number of characters (except for a new line character) in an assembler source line has exceeded 2,047 characters.
Error: Cannot allocate memory. *	Memory allocation by <code>malloc()</code> has failed.
Error: Cannot specify plurality source files. *	More than one source file name is specified in the command line.
Error: Cannot find the dump file. *	A dump file name is not specified even though the <code>-mc17_ext</code> option is specified. Or the specified dump file does not exist.
Error: Cannot find the map file. *	A map file name is not specified even though the <code>-mc17_ext</code> option is specified. Or the specified map file does not exist.
Error: The format of the dump file is invalid. *	The contents in the dump file specified with the <code>-mc17_ext</code> option are invalid.
Error: The format of the map file is invalid. *	The contents of the map file specified with the <code>-mc17_ext</code> option are invalid.
Error: Cannot close the map file. *	The map file specified with the <code>-mc17_ext</code> option cannot be closed after it has been read.
Error: There are too many characters of one line in dump file. *	The number of characters (except for a new line character) in a line of the dump file specified with the <code>-mc17_ext</code> option has exceeded 2,047 characters.
Error: There are too many characters of one line in map file. *	The number of characters (except for a new line character) in a line of the map file specified with the <code>-mc17_ext</code> option has exceeded 2,047 characters.

\* When the source file is assembled through **xgcc**, assembling terminates after displaying the following error message instead of the above message:

```
xgcc: cannot specify -o with -c or -S and multiple compilations
```

Table 8.8.2 Warning messages

Warning message	Description
Warning: Unrecognized .section attribute: want a, w, x	The section attribute is not a, w or x.
Warning: Bignum truncated to AAA bytes	The constant declared (e.g. <code>.long</code> , <code>.int</code> ) exceeds the maximum size. It has been corrected to AAA-byte size. (e.g. <code>0x100000012</code> → <code>0x12</code> )
Warning: Value XXXX truncated to AAA	The constant declared exceeds the maximum value AAA. It has been corrected to AAA. (e.g. <code>.byte 0x100000012</code> → <code>.byte 0xff</code> )

## 8.9 Precautions

- To perform assembly source level debugging with the debugger **gdb**, specify the `--gstabs` assembler option to add the source information to the output object file when assembling the source file.
- Always be sure to use the **xgcc** compiler and/or **as** assembler to add debugging information (`.stab` directive) in the source file and do not use any other method. Also be sure not to correct the debugging information that is output. Corrections could cause the **as**, **ld** or **gdb** to malfunction.
- To prevent errors during linkage, be sure to write the `.section` directive with the `.align` directive to clearly define the section boundary.

Example:

```

        .section .rodata
        .align 2           ; ← Essential
        .long   data1
        .long   data2

```

- Up to 512 characters can be used for a global symbol name and up to 32,000 global symbols can be defined.
- The assembler source, map and dump files have an upper limit of 2,047 characters per line (except for a new line character). An error will occur if there is a line that exceeds the limit.
- More than one source file cannot be specified as the input file of the **as** assembler. An error will occur if specified.
- The output file name specified with the `-o` option must be the same as that of the input source file name (except for the file name extension).
- When the `ext` instructions are written directly to an assembler source to extend basic instructions, they will not be optimized even if the `-mc17_ext` option is specified. When creating an assembler source, use `s*` or `x*` extended instructions for data transfer, arithmetic operation, and branch if a symbol is used as the operand.
- If the branch destination symbol for a relative branch instruction is located in another section, the branch instruction will not be optimized even if the `-mc17_ext` option is specified.

```

Example: .text_A 0x00000000 :
        {
            __START__text_A = . ;
            sub_1.o(.text)
            sub_2.o(.text)
            sub_3.o(.text)
            sub_4.o(.text)
            __END__text_A = . ;
        }
        .text_B 0x00C00000 :
        {
            __START__text_B = . ;
            main.o(.text)
            __END__text_B = . ;
        }

```

In this example, the extended instructions for referencing or branching to a symbol between "main.o" and "sub\_X.o" will not be optimized. Therefore, new sections should not be added in the linker script file (`.lds`) if they are not necessary.

- When a file name is described twice or more in a dump file (source files with the same name (except extension and path) exist), optimization for local symbols using the dump file will not be performed. This is not applied when the uppercase and lowercase letter configuration is different even if the file name is the same. Furthermore, this condition does not affect optimization for global symbols using a map file.



# 9 Linker

This chapter describes the functions of the **ld** linker.

## 9.1 Functions

The **ld** linker is a software that generates executable object files. It provides the following functions:

- Links together multiple object modules including libraries to create one executable object file.
- Resolves external reference from one module to another.
- Relocates relative addresses to absolute addresses.
- Delivers debugging information, such as line numbers and symbol information, in the object file created after linking.
- Capable of outputting link map files.

This linker is based on the gnu linker (ld). For details about the **ld** linker, refer to the documents for the gnu linker. The documents can be acquired from the GNU mirror sites located in various places around the world through Internet, etc.

## 9.2 Input/Output Files

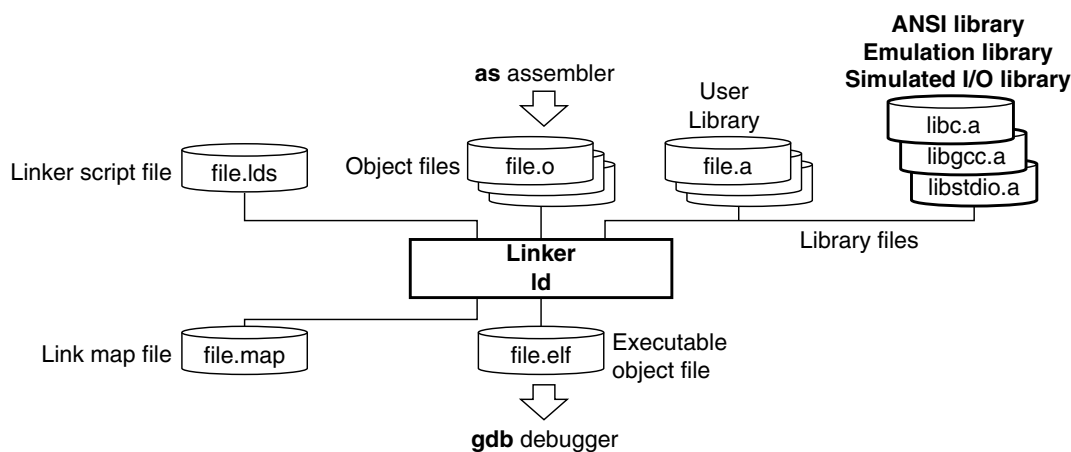


Figure 9.2.1 Flowchart

### 9.2.1 Input Files

#### Object file

File format: Binary file in elf format

File name: `<file name>.o`

Description: Object file of individual modules created by the **as** assembler.

#### Library file

File format: Binary file in library format

File name: `<file name>.a`

Description: ANSI library files, emulation library files and user library files.

#### Linker script file

File format: Text file

File name: `<file name>.lds`

Description: File to specify the start address of each section and other information for linkage.

The **IDE** may be used to create a linker script file.

It is input to the **ld** linker when the `-T` option is specified.

## 9.2.2 Output Files

### Executable object file

File format: Binary file in elf format

File name: `<file name>.elf`

Description: Object file in executable format that can be input in the **gdb** debugger. All the modules comprising one program are linked together in the file, and the absolute addresses that all the codes will be mapped are determined. It also contains the necessary debugging information in elf format.

The default file name is `a.out` when no output file name is specified using the `-o` option.

### Link map file

File format: Text file

File name: `<file name>.map`

Description: Mapping information file showing from which address of a section each input file was mapped.

The file is delivered when the `-M` or `-Map` option is specified.

## 9.3 Starting Method

---

### 9.3.1 Startup Format

To invoke the **ld** linker, use the command shown below.

`ld <options> <file names>`

`<options>` See Section 9.3.2.

`<file names>` Specify one or more object file names and/or one or more library file names.

Example: `ld -o sample.elf boot.o sample.o ..\lib\24bit\libc.a ..\lib\24bit\libgcc.a`

### 9.3.2 Command-line Options

The **ld** linker accepts the gnu linker standard options. The following lists the principal options only. Refer to the gnu linker manual for more information.

**-o** `<file name>`

Function: **Specify output file name**

Explanation: This option is used to specify the name of the object file output by the **ld** linker.

Default: The default output file name is `a.out`.

**-T** `<linker script file name>`

Function: **Read linker script file**

Explanation: Specify this option when loading relocate-information into the **ld** linker using a linker script file.

Default: The default linker script (see Section 9.4.1) is used.

**-M**

**-Map** `<file name>`

Function: **Output link map file**

Explanation: The `-M` option outputs the link map information to `stdio`.

The `-Map` option outputs the link map information to a file.

Default: No link map information is output.

**-N**

Function: **Disable data segment alignment check**

Explanation: When the `-N` option is specified, the linker does not check the alignment of data segments. This option should be used normally. (see Section 9.6, "Precautions".)

Default: The linker checks the alignment of data segments.

When inputting options in the command line, one or more spaces are necessary before and after the option.

Example: `ld -o sample.elf -T sample.lds -N boot.o sample.o ..\lib\24bit\libc.a`

## 9.4 Linkage

### 9.4.1 Default Linker Script

#### Default linker script when the `-T` option is not specified

When the `-T` option is not specified, the `ld` linker uses the default script shown below for linkage.

```
OUTPUT_FORMAT("elf32-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);
SECTIONS
{
    /* section information */
    .bss 0x0 :
    {
        __START_bss = . ;
        *(.bss)
        __END_bss = . ;
    }
    .data __END_bss : AT( __END_rodata )
    {
        __START_data = . ;
        *(.data)
        __END_data = . ;
    }
    .text 0x8000 :
    {
        __START_vector = . ;
        *(.vector)
        __END_vector = . ;
        . = 0x80 ;
        __START_text = . ;
        *(.text)
        __END_text = . ;
    }
    .rodata __END_text :
    {
        __START_rodata = . ;
        *(.rodata)
        __END_rodata = . ;
    }
    /* load address symbols */
    __START_data_lma = LOADADDR( .data );
}
```

In this script, data will be located from address 0 in order of `.bss` and `.data` sections, the vector table, program codes and constant data will be located from address 0x8000.

Figure 9.4.1.1 shows the memory map after linkage.

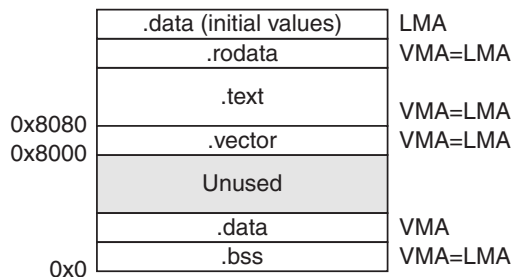


Figure 9.4.1.1 Memory map configured by default script

## 9.4.2 Examples of Linkage

### When virtual and shared sections are used

The following is a sample linker script when virtual and shared sections are used:

```

OUTPUT_FORMAT("elf32-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);

SECTIONS
{
    /* location counter */
    . = 0x0;

    /* section information */
    .bss 0x000000 :
    {
        __START_bss = . ;
        *(.bss) ;
        __END_bss = . ;
    }

    .data __END_bss : AT( __END_text )
    {
        __START_data = . ;
        *(.data) ;
        __END_data = . ;
    }
    __START_data_lma = LOADADDR( .data );

    .text_foo1 __END_data : AT( __START_data_lma+SIZEOF( .data )
    {
        __START_text_foo1 = . ;
        foo1.o(.text) ;
        __END_text_foo1 = . ;
    }
    __START_text_foo1_lma = LOADADDR( .text_foo1 );

    .text_foo2 __END_data : AT( __START_text_foo1_lma+SIZEOF( .text_foo1 )
    {
        __START_text_foo2 = . ;
        foo2.o(.text) ;
        __END_text_foo2 = . ;
    }
    __START_text_foo2_lma = LOADADDR( .text_foo2 );

    .text_foo3 __END_data : AT( __START_text_foo2_lma+SIZEOF( .text_foo2 )
    {
        __START_text_foo3 = . ;
        foo3.o(.text) ;
        __END_text_foo3 = . ;
    }
    __START_text_foo3_lma = LOADADDR( .text_foo3 );

    .rodata 0x008000 :
    {
        __START_rodata = . ;
        *(.rodata) ;
        __END_rodata = . ;
    }

    .text __END_rodata :
    {
        __START_text = . ;
        *(.text) ;
        __END_text = . ;
    }
}

```

The section map is shown in Figure 9.4.2.1.

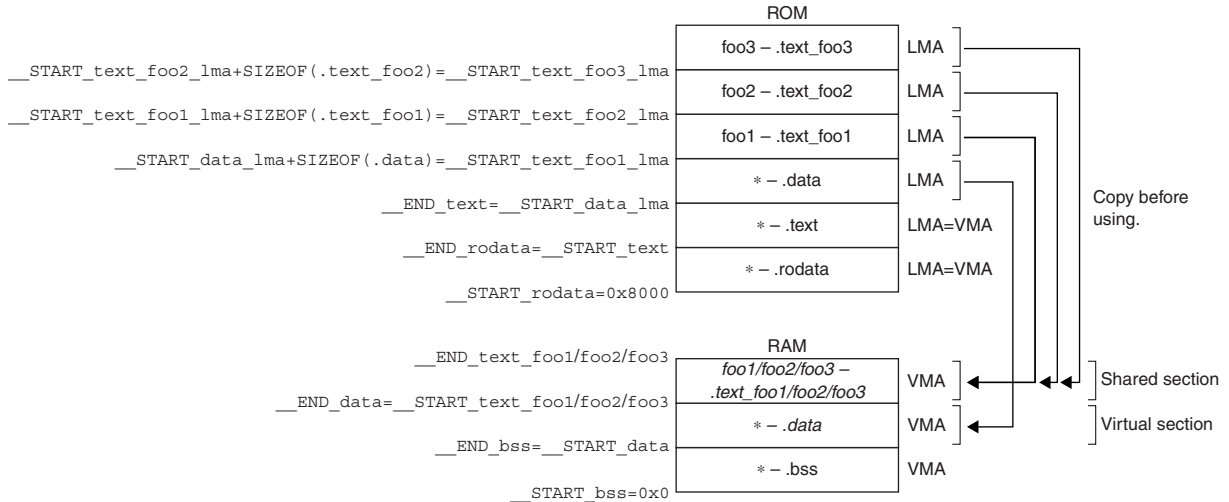


Figure 9.4.2.1 Memory map

The substance of the `.data` section is placed on the LMA in the ROM, and it must be copied to the VMA in the RAM (immediately following the `.bss` section) before it can be used. The `.data` section (VMA) in the RAM is a virtual section that does not exist when the program starts executing. This method should be used for handling variables that have an initial value. In this example, the `.data` sections in all the files are combined into one section.

`.text_foo1` is the `.text` section in the `foo1.o` file. Its actual code is located at the LMA in the ROM and is executed at the VMA in the RAM. Also the `.text_foo2` and `.text_foo3` sections are used similarly and the same VMA is set for these three sections. The RAM area for `.text_foo1/2/3` is a shared section used for executing multiple `.text` sections by replacing the codes. A program cache for high-speed program execution is realized in this method. The `.text` sections in other files than these three files are located in the `.text` section that follows the `.rodata` section (0x8000-) and are executed at the stored address in the ROM.

## 9.5 Error Messages

---

Error messages are displayed/output through the Standard Output (stdout).

In the **ld** linker, the following error messages are added to the standard error messages of the gnu linker:

Table 9.5.1 Error messages

Error message	Description
Error: The offset value of a symbol is over 16bit.	The address of the symbol exceeds the 16-bit address space.
Error: The offset value of a symbol is over 24bit.	The address of the symbol exceeds the 24-bit address space.
Error: Input object file use both 16bit and 24bit address mode.	The object files to be linked contain both files created in 24-bit pointer mode and 16-bit pointer mode.
Error: section <i>XXX</i> is not within 16bit address.	The address of the <i>XXX</i> section exceeds the 16-bit address space.
Error: section <i>XXX</i> is not within 24bit address.	The address of the <i>XXX</i> section exceeds the 24-bit address space.

## 9.6 Precautions

- When the linker is executed, an error message as shown below may appear.

```
ld: test.elf: Not enough room for program header, try linking with -N
```

This error occurs in the alignment check for the data segment. The linker's alignment check can be disabled with the `-N` option, so normally specify the `-N` option when invoking the linker. (The make file generated by the **IDE** contains the linker command line with the `-N` option.)

- The object file names are case-sensitive. It is necessary to specify the exact same file name in the `ld` command line and the linker script file. If the upper/lower case is different, `ld` considers them as two different files.

Example:

Command line

```
ld -T sample.lds -o sample.elf prg1.o prg2.o
```

Linker script file (sample.lds)

```

:
.text 0xc00000:
{
PRG1.o (.text) ← PRG1.o must be changed to prg1.o.
prg2.o (.text)
}
:

```

- Linking files of different sizes with the same function name displays the following message.

```
Warning: size of symbol 'AAA' changed from BBB to CCC in DDD.o
```

```
AAA:      Duplicate function name
BBB, CCC: Size of function
DDD:      File name to be linked
```

If file sizes match, this message is not displayed. When linking files, be careful to avoid specifying the same function name. Take particular care to avoid assigning a function name already included in the library file (`*.a`).

- If `"*"` and individual object file specification coexist in a linker script file as in the example below, the files may not link correctly. If object files do not need to be specified individually, use only `"*"`. With this exception, specify object files individually.

Example:

```

.text 0x00600000 :
{
*(.text) ;
}
.text2 :
{
main.o(.text) ;
}

```

- Linking two or more library files (`*.a`) that contain the same function does not cause an error (no double linkage performed).

Note that an error occurs when two or more object files (`*.o`) that contain the same function are linked.

- If the located address, which is specified by a variable or the result of a calculation with a variable, is higher than the 24-bit limit (`0xfffff`) or lower than `0x0`, the address bits that exceed 24 bits are masked with 0 and no error occurs.

Example: `xadd.a %r0, symbol-5`

If the `symbol` is located at address 0, the specified absolute address is  $0 - 5 = 0xfffffb$  (-5). Therefore, this code will be assembled as `"xadd.a %r0, 0xfffffb"`.

THIS PAGE IS BLANK.



# 10 Debugger

This chapter describes how to use the debugger **gdb**.

## 10.1 Features

The debugger **gdb** is software used to debug a program after loading an elf-format object file created by the linker. This debugger has the following features and functions:

- Can reference various types of data at one time, thanks to a multi-window facility.
- In addition to debugging programs using the ICD Mini (S5U1C17001H) or ICD board, the debugger incorporates a software simulator function for debugging programs on a personal computer.
- Capable of C source and assembly source level debugging.
- Supports C source and assembler level single-stepping functions, in addition to continuous program execution.
- Supports hardware and software PC break functions.
- Can measure program execution time by duration of time or number of cycles.
- Supports a trace function that allows saving of the traced data (in simulator mode).
- Can automatically execute commands using a command file.
- Supports a simulated I/O function that allows Stdin/Stdout evaluation in the debugger.
- Supports the embedded system simulator that allows evaluation of port input/output, SVD and LCD display in the debugger.
- Supports the flash writer function of the ICD Mini (S5U1C17001H).

## 10.2 Input/Output Files

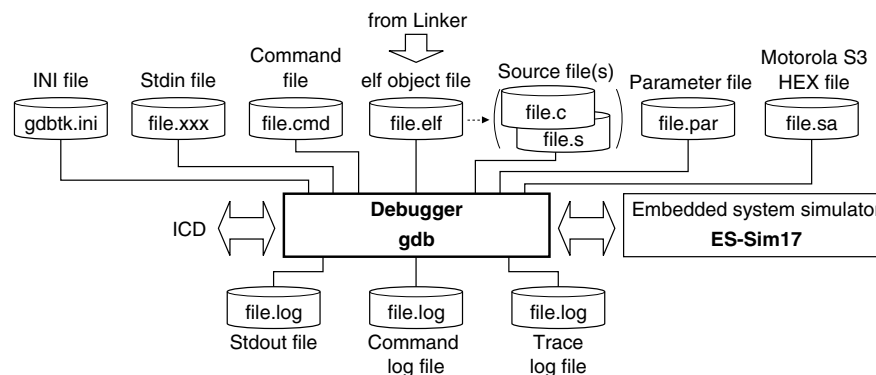


Figure 10.2.1 Flowchart

### 10.2.1 Input Files

#### Parameter file

File format: Text file

File name: `<filename>.par`

Description: This file has recorded in it the contents needed to set the memory map information for the debugger. The [Project > Properties > GNU17 Parameter Settings] dialog box of the **IDE** displayed by selecting Project and then Properties may be used to create parameter files. For details about parameter files, see Section 10.8, "Parameter Files".

#### INI file

File format: Text file

File name: `gdbtk.ini`

Description: This file is loaded in the debugger at startup to initialize the debugger. Closing the debugger updates such setup contents as the window position and display color. To revert the various debugger settings to the original factory settings, copy the original `gdbtk.ini` from the `\gnu17\utility\share` directory to the `gdb.exe` directory (`\gnu17`) over the existing file there.

### Object file

File format: elf format binary file

File name: `<filename>.elf`

Description: This is the elf format absolute object file created by the linker **ld**. This file is loaded in the debugger by using the `file` and `load` commands. Source display and symbolic debugging are made possible by loading an object file that contains debug information.

### Source files

File format: Text file

File name: `<filename>.c` (C source), `<filename>.s` (assembly source)

Description: These are source files for the object file above, loaded in the debugger to generate source display.

### Command file

File format: Text file

File name: `<filename>.cmd`

Description: This file contains a description of debugging commands to be executed successively. By writing a series of frequently used commands in a file, you can save the time and labor required for entering commands from the keyboard each time. This file is loaded in the debugger and executed by the startup option `-x` or the `source` command.

### ROM data HEX file

File format: Motorola S3 format HEX file

File name: `<filename>.sa`

Description: This file cannot be used for source-level debugging because it does not contain debug information. Motorola S3 format files are needed when using the flash writer function. This file is loaded in the debugger by the `c17 fwlp` or `c17 fwld` command.

Files in this format may be created from elf format files output from the linker **ld** by converting them with the file format conversion tool **objcopy**.

**objcopy** format) `objcopy -O srec --srec-forceS3 <filename>.elf <filename>.sa`

Example: `objcopy -O srec --srec-forceS3 sample.elf sample.sa`

### Input simulation data file

File format: Text file

File name: Any file name

Description: This file is loaded in the debugger by the simulated input function. The `c17 stdin` command is used to specify the file name and other information.

## 10.2.2 Output Files

### Trace information file

File format: Text file

File name: Any file name

Description: This file contains trace results output by the debugger. The `c17 tm` command is used to specify a file name and other information for trace results to be output.

### Log file

File format: Text file

File name: Any file name

Description: The commands executed and execution results are output to this file. The `c17 log` command is used to specify a file name and other information for logs to be output.

### Output simulation data file

File format: Text file

File name: Any file name

Description: This file is created by the simulated output function. The `c17 stdout` command is used to specify a file name and other information.

## 10.3 Starting the Debugger

### 10.3.1 Startup Format

#### General command line format

```
gdb [<startup option>]
```

The brackets [ ] denote that the specification can be omitted.

### 10.3.2 Startup Options

The debugger has seven available startup options. For more information on how to select in the **IDE**, refer to Section 5.8.3, "Starting Up the Debugger."

**--command=<command filename>**

**-x <command filename>**

Function: **Specifies a command file**

Explanation: When this option is specified, the debugger loads the specified command file at startup and executes the commands written in the file.

**--c17\_cmw=<wait in seconds>**

Function: **Specifies a time interval at which to execute commands in a command file**

Explanation: When the debugger executes a command file as specified by the **-x** or **--command** option or **source** command, this option inserts a wait time between each command by a specified duration in seconds. The wait time can be specified from 1 to 256 seconds. If any other value is specified, a 1-second wait time is assumed.

**--cd=<directory path string>**

Function: **Changes the current directory**

Explanation: When this option is specified, the debugger sets the specified path for the current directory at startup. If this option is omitted, the directory written in the `gdbtk.ini` file (or directory containing **gdb.exe**, if `gdbtk.ini` not found) is assumed.

**--directory=<directory path string>**

Function: **Changes the source file directory**

Explanation: This option can be used to specify the directory containing the source files. Multiple instances of this option can be specified.

**--c17\_euc**

Function: **Displays the EUC code**

Explanation: When this option is specified, the compiler assumes that kanji code is EUC when it displays code in the source window.

By default, kanji code is assumed to be Shift JIS, and EUC codes are displayed as a blank.

**--c17\_double\_starting**

Function: **Enables double starting**

Explanation: This option enables launching a second instance of the debugger in one PC that is disabled by default.

When entering options on the command line, insert one or more spaces to delimit each option.

Example: `c:\EPSON\gnu17\gdb -x sample.cmd --cd=/cygdrive/d/test/sample`

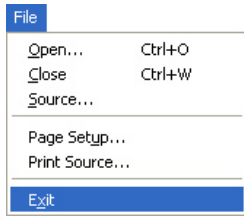
**Notes:** • An error message as shown below will be displayed if an unknown option is specified.

Error message: `... gdb : unrecognized option 'XXXXXX'`

- A second instance of the debugger cannot be run in one PC by default.

### 10.3.3 Quitting the Debugger

To quit the debugger, select [Exit] from the [File] menu in the [Source] window.



[File] menu

Otherwise, you can enter the `quit (q)` command in the [Console] window to close the debugger.

(gdb) `q`

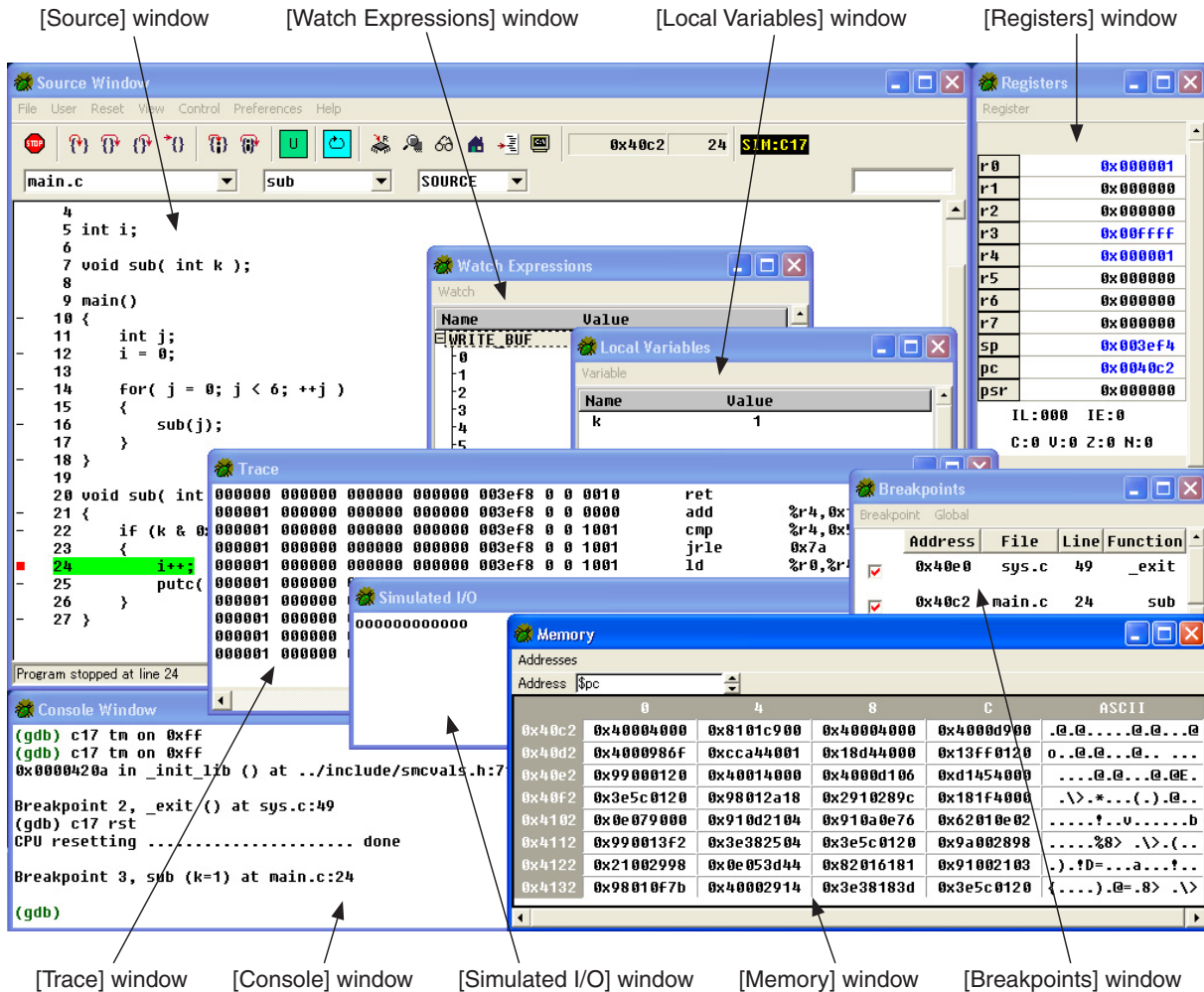
**Note:** When using an ICD to debug a program (in ICD Mini mode), always be sure to close the debugger before turning off power to the ICD. Should you turn off power to the ICD while running the debugger, you will be unable to reconnect it. In such case, you need to restart your computer.

## 10.4 Windows

This section describes the types of windows used in the debugger **gdb**.

### 10.4.1 Basic Window Configuration

The debugger windows basically are configured as shown below.



At debugger startup, the `gdbtk.ini` file is loaded in the debugger, with the [Source], [Console], [Registers] and [Memory] windows opening by default. If the `gdbtk.ini` file does not exist, the debugger creates a `gdbtk.ini` file with the default settings at startup, and opens these four windows. To modify the startup window configuration, correct the [window] section in the `gdbtk.ini` file. To revert to its original factory settings after modifying the window configuration, copy the original `gdbtk.ini` from the `\gnu17\utility\share` directory to the `gdb.exe` directory (`\gnu17`) over the existing file there.

All windows (except the [Source] window) can be opened manually by using the [View] menu or tool bar buttons in the [Source] window. Clicking the [Close] button also closes windows.

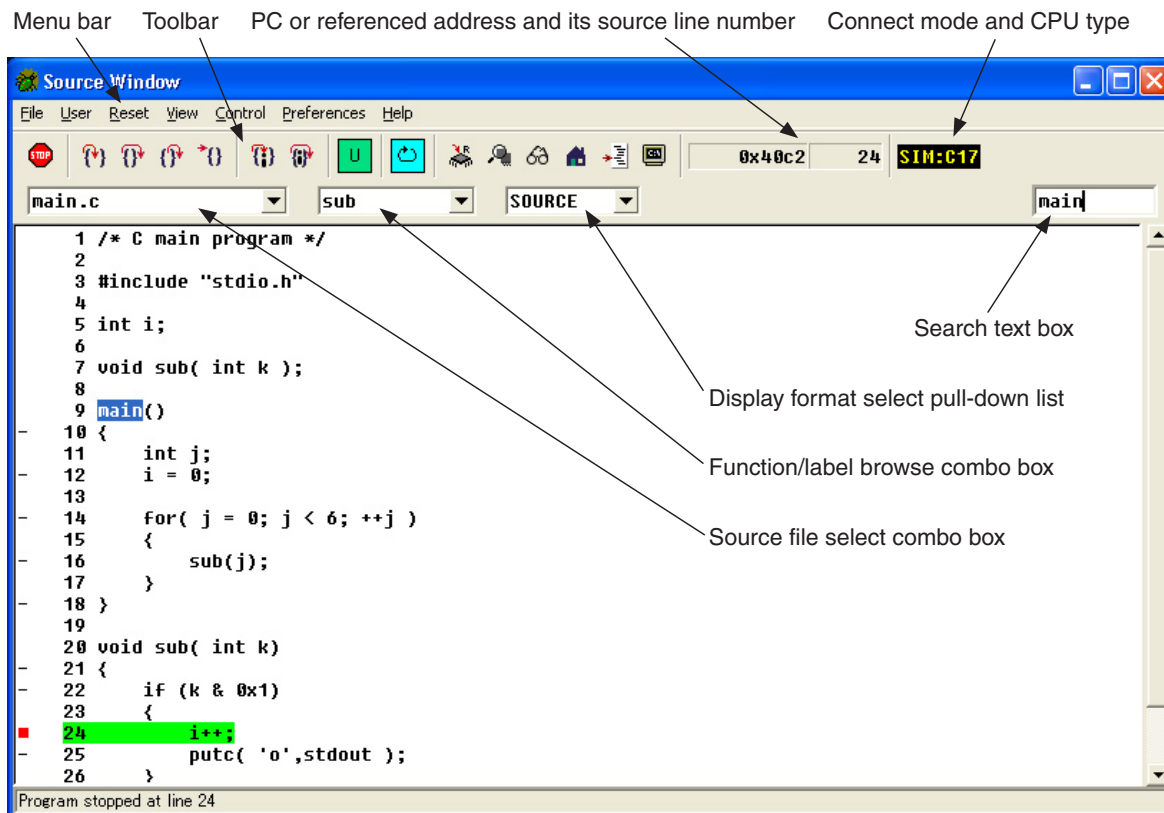
Closing the [Source] window also closes the debugger.

All these windows can be moved, zoomed in or out, minimized, or maximized.

## 10.4.2 [Source] Window

The [Source] window is the main window of the debugger. It has all the menus and toolbar buttons generally associated with a debugger. The display area of this window is designed not just to display programs, but also to enable software PC breakpoints to be set and cleared, for example.

### Window configuration



### Menus

#### [File] menu

File	
Open...	Ctrl+O
Close	Ctrl+W
Source...	
Page Setup...	
Print Source...	
Exit	

#### Open...

Displays a dialog box for selecting an elf format object file. When a file is selected, the debugger executes the `file` command to load debug information.

#### Close

Closes the currently open elf format object file. The debug information and contents displayed in the window are cleared.

#### Source...

Displays a dialog box for selecting a command file. When a file is selected, the debugger executes the `source` command using the selected file.

#### Page Setup...

Selects the paper size and related settings, and sets up the page on which to print the sources.

#### Print Source...

Prints the currently displayed source in exactly the same form as displayed on the screen.

#### Exit

Quits the debugger.

**[User] menu**

User
User Command

**User Command**

Executes the command file `\gnu17\userdefine.gdb` that can be edited by the user.

Contents of `userdefine.gdb` at shipment

```
#Edit user command
#c17 rst          (No command executed)
```

**[Reset] menu**

Reset
Reset

**Reset**

Executes the command file `\gnu17\reset.gdb` that can be edited by the user.

Contents of `reset.gdb` at shipment

```
c17 rst          (Resets the CPU)
```

**[View] menu**

View	
Registers	Ctrl+R
Memory	Ctrl+M
Watch Expressions	Ctrl+T
Local Variables	Ctrl+L
Breakpoints	Ctrl+B
Console	Ctrl+N
SimI/O	Ctrl+I
Trace	Ctrl+A

**Registers**

Opens the [Registers] window. If already open, the window moves to the front of the screen.

**Memory**

Opens the [Memory] window. If already open, the window moves to the front of the screen.

**Watch Expressions**

Opens the [Watch Expressions] window. If already open, the window moves to the front of the screen.

**Local Variables**

Opens the [Local Variables] window. If already open, the window moves to the front of the screen.

**Breakpoints**

Opens the [Breakpoints] window. If already open, the window moves to the front of the screen.

**Console**

Opens the [Console] window. If already open, the window moves to the front of the screen.

**SimI/O**

Opens the [Simulated I/O] window. If already open, the window moves to the front of the screen.

**Trace**

Opens the [Trace] window. If already open, the window moves to the front of the screen.

**[Control] menu**

Control	
Step	S
Next	N
Finish	F
Continue	C
Step Asm Inst	S
Next Asm Inst	N

**Step**

Issues the `step` command to execute the target program starting at one source line from the current PC address.

**Next**

Issues the `next` command to execute the target program starting at one source line from the current PC address. In this case, all function and subroutine calls (including called functions and subroutines) are executed as one step.

**Finish**

Issues the `finish` command to execute the target program from the current PC address. The program is made to stop upon returning from the current function to its caller function or routine.

**Continue**

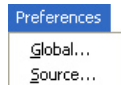
Issues the `continue` command to continuously execute the target program from the current PC address.

**Step Asm Inst**

Issues the `stepi` command to execute the target program starting at one mnemonic instruction from the current PC address.

**Next Asm Inst**

Issues the `nexti` command to execute the target program starting at one mnemonic instruction from the current PC address. In this case, all subroutine calls (including called subroutines) are executed as one step.

**[Preferences] menu****Global...**

Displays the [Global Preferences] dialog box used to select the fonts. For details about the content set in this dialog box, see "Changing the basic settings" later in this section.

**Source...**

Displays the [Source Preferences] dialog box used to select the source display color, tab width or the like. For details about the content set in this dialog box, see "Changing the basic settings" later in this section.

**[Help] menu****About GDB...**

Shows the debugger version and copyright notes in a dialog box.

**Toolbar buttons****[Stop] button**

Forcibly stops target program execution. When the CPU is placed in standby mode (HALT or SLEEP) by running the target program or the program enters an endless loop, this button may be used to break program execution.

**[Step] button**

Issues the `step` command to execute the target program starting at one source line from the current PC address.

**[Next] button**

Issues the `next` command to execute the target program starting at one source line from the current PC address. In this case, all function and subroutine calls (including called functions and subroutines) are executed as one step.

**[Finish] button**

Issues the `finish` command to execute the target program from the current PC address. The program is made to stop upon returning from the current function to a higher-level function or routine.

**[Continue] button**

Issues the `continue` command to continuously execute the target program from the current PC address.

**[Step Asm Inst] button**

Issues the `stepi` command to execute the target program starting at one mnemonic instruction from the current PC address.

**[Next Asm Inst] button**

Issues the `nexti` command to execute the target program starting at one mnemonic instruction from the current PC address. In this case, all subroutine calls, including called subroutines, are executed as one step.



**[User Command] button**

Executes the command file `\gnu17\userdefine.gdb` that can be edited by the user. The debugger uses the image file shown below to display this button. This button can be customized by replacing the contents in this file with a user created  $24 \times 24$ -dot button image.  
`\gnu17\utility\share\gdbtcl\images2\userdefine.gif`

**[Reset] button**

Executes the command file `\gnu17\reset.gdb` that can be edited by the user.

**[Registers] button**

Opens the [Registers] window. If already open, the window moves to the front of the screen.

**[Memory] button**

Opens the [Memory] window. If already open, the window moves to the front of the screen.

**[Watch Expressions] button**

Opens the [Watch Expressions] window. If already open, the window moves to the front of the screen.

**[Local Variables] button**

Opens the [Local Variables] window. If already open, the window moves to the front of the screen.

**[Breakpoints] button**

Opens the [Breakpoints] window. If already open, the window moves to the front of the screen.

**[Console] button**

Opens the [Console] window. If already open, the window moves to the front of the screen.

**Content displayed in window****Program display**

The [Source] window displays the contents of the loaded program. Only one source file at a time can be displayed in this window. If necessary, multiple instances of the [Source] window can be opened, with separate source files displayed in each.

One of four display forms can be selected from the pull-down list.



**Source display** (with [SOURCE] selected from pull-down list)

The window shows source line numbers and source codes. This is the default display format selected at debugger startup.

```

1 /* C main program */
2
3 #include "stdio.h"
4
5 int i;
6
7 void sub( int k );
8
9 main()
10 {
11     int j;
12     i = 0;
13
14     for( j = 0; j < 6; ++j )
15     {
16         sub(j);
17     }
18 }
19
20 void sub( int k)
21 {
22     if (k & 0x1)
23     {
24         i++;
25         putchar( 'o',stdout );
26     }

```

Program stopped at line 14

**Assembler display** (with [ASSEMBLY] selected from pull-down list)

The window shows addresses, labels, and basic and extended assembler instructions.

```

0x40a0 <main>: ld.a -[%sp],%r4 ld.a -[%sp],%r4
0x40a2 <main+2>: ld %r2,0x0 ld %r2,0x0 <i>
0x40a4 <main+4>: ext 0x0
0x40a6 <main+6>: ext 0x0
0x40a8 <main+8>: ld [0x0],%r2 xld [0x0],%r2 <i>
0x40aa <main+10>: ld %r4,%r2 ld %r4,%r2
0x40ac <main+12>: ld %r0,%r4 ld %r0,%r4
0x40ae <main+14>: ext 0x0
0x40b0 <main+16>: call 0x5 xcall 0x5 (0x0040BC) <sub:
0x40b2 <main+18>: add %r4,0x1 add %r4,0x1
0x40b4 <main+20>: cmp %r4,0x5 cmp %r4,0x5
0x40b6 <main+22>: jrle 0x7a jrle 0x7a (0x0040AC)
0x40b8 <main+24>: ld.a %r4,[%sp]+ ld.a %r4,[%sp]+
0x40ba <main+26>: ret ret

```

Program stopped at line 14

**Note:** The object codes in the source file are disassembled before being displayed here. Therefore, the content displayed here does not always reflect that of the assembly source.

**Mixed display** (with [MIXED] selected from pull-down list)

The window provides a mixed display of source and assembler, with the corresponding assembler display inserted every source line.

```

Source Window
File User Reset View Control Preferences Help
0x40aa 14 SIM:C17
main.c main MIXED
10 {
0x40a0 <main>:          ld.a  -[%sp],%r4      ld.a  -[%sp],%r4
11     int j;
12     i = 0;
0x40a2 <main+2>:        ld    %r2,0x0        ld    %r2,0x0 <i>
0x40a4 <main+4>:        ext   0x0            ld    %r2,0x0 <i>
0x40a6 <main+6>:        ext   0x0            ld    %r2,0x0 <i>
0x40a8 <main+8>:        ld    [%x0],%r2      xld   [%x0],%r2 <i>
13
14     for( j = 0; j < 6; ++j )
0x40aa <main+10>:       ld    %r4,%r2        ld    %r4,%r2
0x40b2 <main+18>:       add   %r4,0x1        add   %r4,0x1
0x40b4 <main+20>:       cmp   %r4,0x5        cmp   %r4,0x5
0x40b6 <main+22>:       jrle  0x7a           jrle  0x7a (0x00408C)
15     {
16         sub(j);
0x40ac <main+12>:       ld    %r0,%r4        ld    %r0,%r4
0x40ae <main+14>:       ext   0x0            ld    %r0,%r4
0x40b0 <main+16>:       call  0x5            xcall 0x5 (0x00408C) <sub:
17     }
18 }
0x40b8 <main+24>:       ld.a  %r4,[%sp]+     ld.a  %r4,[%sp]+
0x40ba <main+26>:       ret
Program stopped at line 14

```

**Split display** (with [SRC+ASM] selected from pull-down list)

The window is split into upper and lower halves, with the upper area showing source lines, and the lower area showing assembler display.

```

Source Window
File User Reset View Control Preferences Help
0x40aa 14 SIM:C17
main.c main SRC+ASM
1 /* C main program */
2
3 #include "stdio.h"
4
5 int i;
6
7 void sub( int k );
8
9 main()
10 {
11     int j;
12     i = 0;
13
0x40a0 <main>:          ld.a  -[%sp],%r4      ld.a  -[%sp],%r4
0x40a2 <main+2>:        ld    %r2,0x0        ld    %r2,0x0 <i>
0x40a4 <main+4>:        ext   0x0            ld    %r2,0x0 <i>
0x40a6 <main+6>:        ext   0x0            ld    %r2,0x0 <i>
0x40a8 <main+8>:        ld    [%x0],%r2      xld   [%x0],%r2 <i>
0x40aa <main+10>:       ld    %r4,%r2        ld    %r4,%r2
0x40ac <main+12>:       ld    %r0,%r4        ld    %r0,%r4
0x40ae <main+14>:       ext   0x0            ld    %r0,%r4
0x40b0 <main+16>:       call  0x5            xcall 0x5 (0x00408C) <s
0x40b2 <main+18>:       add   %r4,0x1        add   %r4,0x1
0x40b4 <main+20>:       cmp   %r4,0x5        cmp   %r4,0x5
Program stopped at line 14

```

The relative sizes of upper and lower areas can be changed by dragging the borders of both areas up or down.

### Source display conditions

Source line numbers and source codes can only be displayed when an elf object file containing debug information for source display is loaded.

For C sources, only the sources in a file compiled after specifying C compiler option `-gstabs` are displayed.

For assembly sources, only the sources in a file assembled after specifying assembler option `--gstabs` are displayed.

If these items of information are not included or the corresponding source file cannot be found, the program is forcibly displayed in assembler format.

### Current PC (program counter) display

Once a program is executed, the current PC address line (i.e., next line to be executed) is highlighted in green (default). The highlighting color can be changed in the [Source Preferences] dialog box that appears when [Source...] is selected from the [Preferences] menu.

### Breakpoint display

The bar "-" displayed at the beginning of a source line or address denotes that a PC breakpoint may be set there. (In assembler display, breaks cannot be set at some addresses.)

The source lines or addresses marked with ■ instead of the bar "-" indicate that PC breakpoints are set there. The color of the mark represents the type or status of the breakpoint.

■ Red: The breakpoint set here is a normal software PC breakpoint and enabled.

■ Orange: The breakpoint set here is a temporary software PC breakpoint (cleared upon a program break) and enabled.

■ Blue: The breakpoint set here is a hardware PC breakpoint and enabled.

■ Black: The breakpoint set here is a PC breakpoint and currently disabled.

The color of mark ■ indicating that the PC breakpoint is enabled (by default, red, orange, or blue) can be changed in the [Source Preferences] dialog box that appears when [Source...] is selected from the [Preferences] menu.

### Symbol value display

When the mouse pointer is positioned over a symbol name (also possible in a comment, etc., when the symbol name constitutes a word), the value of that symbol is displayed.

```

- 10      i = 0;
- 11      for (j=0 ; ; j++)
- 12      {
- 13      sub(j)
- 14      }

```

When the mouse pointer is placed over a variable name, the value of that variable is displayed.

However, symbol values are displayed for only symbols belonging to the current stack frame (function), and not for those in other functions.

This function can be disabled in the [Source Preferences] dialog box that appears when [Source...] is selected from the [Preferences] menu.

### Runtime program display

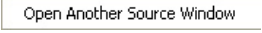
While the program is being executed, the contents displayed in the window are not updated. Instead, the source file or program display positions are updated so that the line at the current PC address is displayed in the window after a break. At the same time, the current PC address and its source line number are displayed in the text box provided to the right of the toolbar buttons.

## Operation

### Opening and closing the window

The [Source] window opens automatically at debugger startup.

If necessary, multiple instances of the [Source] window can be opened to display two or more source files simultaneously. To open another [Source] window, right-click somewhere in the current [Source] window and choose [Open Another Source Window] from the ensuing popup menu.



At least one [Source] window must remain open. When all [Source] windows are closed, the debugger enters shutdown processing. The windows can be minimized, however.

The [Source] window also has a function to open other windows. Choose the window you wish to open from the [View] menu or click the corresponding toolbar button. See the menus and toolbar buttons described above.

### Changing display format

Any display format can be selected from the pull-down list as explained earlier. When you select a different display format, the contents displayed in the window change immediately. However, you cannot change the display format of a source file whose current PC or browse (function/label search) is not highlighted to [ASSEMBLY].

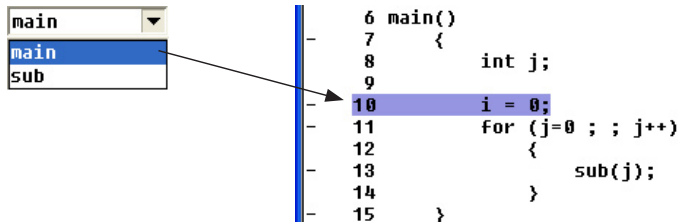
### Selecting between source files

Select the source file name you wish to display from the list in the source file select combo box or enter the source file name from the keyboard.



### Function/label browse

Enter a function name or assembly source label in the function/label browse combo box to update the contents displayed in the window to the specified position. Moreover, the line of instruction on the first source line or at the label position to be executed in that function is highlighted (by default, in cyan).



The background color of this highlighting can be changed in the [Source Preferences] dialog box that appears when [Source...] is selected from the [Preferences] menu.

Because the function or label names defined in the source file displayed are also listed in the combo box, you can choose one from the list and jump to that position.

This browse and highlighting does not affect the PC address or program execution.

### Text search

The text box provided to the far right of the window toolbar is used for text search. From the keyboard, enter the string you wish to search for in this text box, then press the [Enter] key. The specified string is then searched from among the currently displayed contents and when an occurrence of the search string is found, its position is displayed in inverse video. By placing the cursor in the text box, occurrences of the same string can be searched for one after another by simply pressing the [Enter] key.

Search is performed on all text included in the currently displayed contents. Therefore, you can specify disassembled codes, addresses, or line numbers, in addition to those written in the source file. Alphabetic characters are case-sensitive.

### Setting and clearing PC breakpoints

Software PC breakpoints can be set and cleared in the window.

First, choose the source file in which you wish to set breakpoints and display it in the window. To set breakpoints at source lines, choose display format [SOURCE]. To set breakpoints at addresses, choose display format [ASSEMBLY] or [MIXED].

### Using the mouse to set and clear breakpoints

Lines where you can set breakpoints are marked by a bar "-" at the beginning of each line.

Move the cursor near the beginning of a line where you wish to set a breakpoint. The mouse will change shape to a circle (white ○). Click the mouse button there. The "-" mark at the beginning of the line will change to ■ (by default, in red), indicating that a normal software PC breakpoint has been set and remains effective no matter how many times the program is made to break there (by a hit).

Conversely, clicking ■ returns the "-" mark, indicating that the breakpoint you have set is cleared.

This method can only be used to set normal software PC breakpoints. To clear breakpoints, this method can be used for temporary software PC breakpoints (orange ■) and hardware PC breakpoints (blue ■), as well as set PC breakpoints that are currently disabled (black ■).

```

- 10      i = 0;
■ 11      for (j=0 ; ; j++)
12      {

```

### Using a popup menu to set breakpoints at lines or addresses

Right-click near "-" at the beginning of a line to display the popup menu shown below.



Choose [Set Breakpoint] from this menu. The "-" at the beginning of the line will change to ■ (by default, in red), indicating that a normal software PC breakpoint has been set.

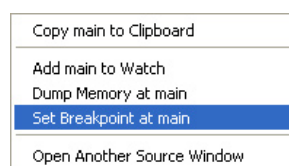
When you choose [Set Temporary Breakpoint], the "-" at the beginning of the line changes to ■ (by default, in orange), indicating that a temporary software PC breakpoint has been set at that line, and one that is only effective for one break hit.

When you choose [Set Hardware Breakpoint], the "-" at the beginning of the line changes to ■ (by default, in blue), indicating that a hardware PC breakpoint has been set.

### Using a popup menu to set breakpoints at beginning of functions

This method is only useful for display format [SOURCE].

Place the cursor over a function or label name, then right-click to display the popup menu shown below.



Choose [Set Breakpoint at <symbol>] from this menu to set a normal software PC breakpoint at the line of instruction on the first source line or at the label position to be executed within that function. The mark at the beginning of the line changes to ■ (by default, in red).

Clicking function call or subroutine call symbols also displays the popup menu, and not just at the beginning of functions.

- Notes:**
- Software PC breakpoints (including temporary breaks) can be set at up to 200 locations. Hardware PC breakpoints can be set at one location. Should this limit be exceeded, including breakpoints set by command input, an error is assumed.
  - Software PC breakpoints cannot be set at addresses in external ROM.
  - PC breakpoints cannot be set at the lines listed below. Otherwise, an error is assumed.
    - Lines of extended instructions, except the `ext` instruction at the beginning
    - Lines of delay instructions (lines next to delayed branch instructions)
  - For other precautions on setting PC breakpoints, see the description of `break`, `tbreak`, `hbreak` and `thbreak` commands.

### Executing the program up to a specified position

Right-click near the "-" mark at the beginning of a line to display the popup menu shown below.

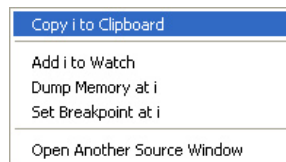


Choose [Jump to Here] from this menu to start running the target program from the current PC address, then stopping it at this line. (This is not effective unless the program processes this line.)

**Note:** The debugger sets a temporary software PC breakpoint to stop the program. Therefore, the precautions on setting software PC breakpoints described above also apply in this case.

### Copying symbols

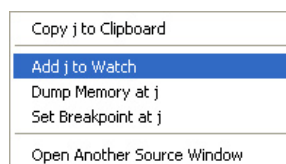
Variable names and function names can be copied to the clipboard using a popup menu. To use this function, change the display format to [SOURCE]. Place the cursor over a variable name, function name, or label, and right-click. This displays the popup menu shown below.



Select [Copy <symbol> to Clipboard] from this menu to copy the symbol name to the clipboard.

### Registering symbols in the watch list

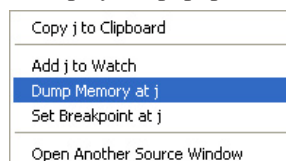
The global symbols you wish to register in the [Watch Expressions] window can be selected from the [Source] window. To use this function, choose display format [SOURCE]. Place the cursor over a global variable name, then right-click to display the popup menu shown below.



Choose [Add <symbol> to Watch] from this menu to register the selected symbol in the [Watch Expressions] window, allowing you to monitor the contents of the variable.

### Symbol-specified memory dump

Choose a symbol in the [Source] window to display a new [Memory] window, with the contents displayed there starting from the address indicated by the value of the selected symbol. To use this function, choose display format [SOURCE]. Place the cursor over a symbol whose memory contents you wish to dump, then right-click to display the popup menu shown below.



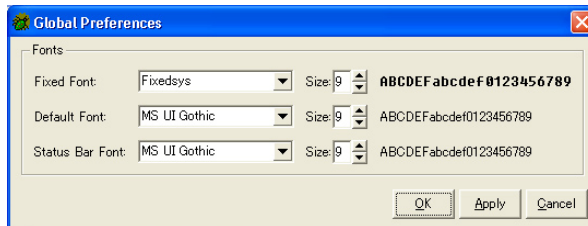
Choose [Dump Memory at <symbol>] from this menu to display a new [Memory] window, showing the contents of memory beginning with the selected symbol value.

**Note:** The address from which the debugger starts displaying the contents of memory is that of the symbol value, and not the address assigned to the selected symbol. Therefore, this is effective for displaying pointer variables, etc.

## Changing the basic settings

### Changing fonts

The display fonts and other basic settings of sources can be changed in the [Global Preferences] dialog box that appears when [Global...] is selected from the [Preferences] menu.



#### Fixed Font

Select a fixed-pitch font to display information in the Source or other windows.

#### Default Font

Select a proportional font to display characters in dialog boxes and in buttons, combo boxes in the [Watch Expressions] and [Memory] windows.

#### Status Bar Font

Select the font for status bar characters.

You can select a font and font size in each of the preceding cases. An example of the selected font is displayed to the right side of each item.

#### [Apply] button

Changes the settings made in the dialog box without closing with the dialog box.

#### [OK] button

Changes settings made in the dialog box and closes the dialog box.

#### [Cancel] button

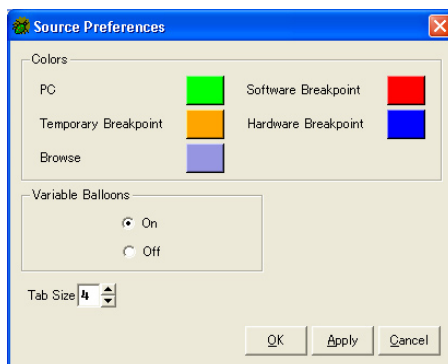
Closes the dialog box without changing the settings you made there. (Note that settings already changed by [Apply] cannot be undone.)

Clicking [Apply] or closing this dialog box updates the contents displayed in the [Source] and other windows.

**Note:** Depending on the font selected, the font size may not be changed, for example, when the Fixed Sys font with a size other than 14 points is selected.

### Changing colors, etc.

The display colors and other basic settings of sources can be changed in the [Source Preferences] dialog box that appears when [Source...] is selected from the [Preferences] menu.



#### Colors

Click a sample color to display the [Choose color] dialog box, allowing you to change the color of an item as desired. Colors of the following items can be changed:

PC: Highlighting of current PC

(by default, R = 0, G = 255 and B = 0)

Browse: Highlighting of referenced function/label line

(by default, R = 149, G = 149, and B = 226)

Software Breakpoint:

Symbols indicating normal software PC breakpoints

(by default, R = 255, G = 0, and B = 0)

Temporary Breakpoint: Symbols indicating temporary software PC breakpoints

(by default, R = 255, G = 165, and B = 0)

Hardware Breakpoint: Symbols indicating hardware PC breakpoints

(by default, R = 0, G = 0, and B = 255)



**Variable Balloons**

Turn on or off the function to display symbol values in a balloon. By default, this function is turned on.

**Tab Size**

Specifies the width of tab stops used in source display by using a number of characters. By default, the tab size is set to 4 characters.

**[Apply] button**

Changes the settings made in the dialog box without closing with the dialog box.

**[OK] button**

Changes settings made in the dialog box and closes the dialog box.

**[Cancel] button**

Closes the dialog box without changing the settings you made there. (Note that settings already changed by [Apply] cannot be undone.)

Clicking [Apply] or closing this dialog box updates the contents displayed in the [Source] and other windows.

**Command execution from menus or toolbar buttons**

The following commands can be executed from menus or toolbar buttons:

<code>c17 rst</code> command:	[Reset] on [Reset] menu, [Reset] button *1
<code>step</code> command:	[Step] on [Control] menu, [Step] button *3
<code>stepi</code> command:	[Step Asm Inst] on [Control] menu, [Step Asm Inst] button *3
<code>next</code> command:	[Next] on [Control] menu, [Next] button *3
<code>nexti</code> command:	[Next Asm Inst] on [Control] menu, [Next Asm Inst] button *3
<code>finish</code> command:	[Finish] on [Control] menu, [Finish] button
<code>continue</code> command:	[Continue] on [Control] menu, [Continue] button *4
User defined command:	[User command] on [User] menu, [User Command] button *2

\*1 The [Reset] command in the [Reset] menu and the [Reset] button execute the command file `\gnu17\reset.gdb` (the `c17 rst` command only is included at shipment).

\*2 The [User command] command in the [User] menu and the [User Command] button execute the command file `\gnu17\userdefine.gdb` (no executable command is included at shipment).

\*3 The `step`, `stepi`, `next` and `nexti` commands are executed for only one step from the current PC address.

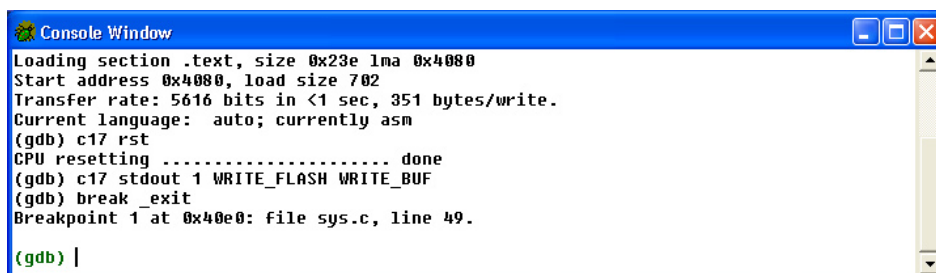
\*4 The `continue` command is executed continuously, without specifying the number of invalid breaks.

**Limitations**

- The contents displayed in the [Source] window cannot be edited from the keyboard, by copying text, or using any other means.
- The contents displayed in the [Source] window do not change even when a command is used to correct the contents of memory in the program area.

### 10.4.3 [Console] Window

The [Console] window is used to execute commands and display execution results.



```

Console Window
Loading section .text, size 0x23e 1ma 0x4080
Start address 0x4080, load size 702
Transfer rate: 5616 bits in <1 sec, 351 bytes/write.
Current language: auto; currently asm
(gdb) c17 rst
CPU resetting ..... done
(gdb) c17 stdout 1 WRITE_FLASH WRITE_BUF
(gdb) break _exit
Breakpoint 1 at 0x40e0: file sys.c, line 49.
(gdb) |

```

#### Displayed contents

When the window is ready for command input, the prompt shown below appears.

```
(gdb) |
```

When you enter a command to execute, the result of command execution is displayed (for only commands with a result output function). For details about the command execution results displayed in this window, see the description of each command later in this manual.

#### Operation

##### Opening/closing the window

The [Console] window opens automatically at debugger startup (unless the `gdbtk.ini` file is modified).

Clicking the [Close] button closes this window. The window can also be minimized.

To reopen the window, choose [Console] from the [View] menu in the [Source] window or click the [Console] button.



[Console] button

Only one instance of the [Console] window can be open at one time. Any attempt to open the [Console] window while already open activates a new [Console] window, which moves to the front of the screen.

##### Command input

In the [Console] window, you can enter and execute any debugging command.

The [Console] window has prompt "`(gdb)`" displayed at the last line, ready to accept command input from the keyboard.

When any other window is selected, click somewhere in the [Console] window. A cursor will start flashing behind the prompt, indicating that the window is ready for command input.

##### Command history

Commands that have thus been executed can be displayed by using the arrow keys ( $\uparrow$ ,  $\downarrow$ ), and can be reexecuted. Simply pressing the [Enter] key reexecutes the previously executed command.

For details on how to enter commands, see Section 10.5.1, "Entering Commands From the Keyboard".

##### Paste

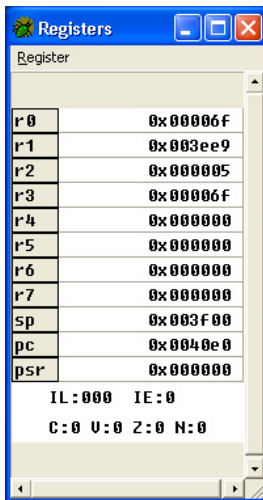
Pressing the [Ctrl] + [V] key combination pastes the copied string at the cursor position in the [Console] window.

##### Limitations

Up to 256 lines can be scrolled in the window. The contents displayed or entered in the window exceeding this limit are deleted.

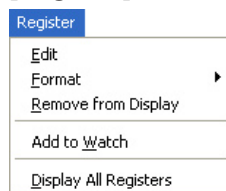
## 10.4.4 [Registers] Window

The [Registers] window is used to display and correct CPU register values.



### Menus

#### [Register] menu



#### Edit

Places the register selected in the window into edit mode. The register value can be changed while in this mode. You cannot choose this command unless a register is selected in the window.

#### Format

Selects the display format for the register selected in the window.

.Hex	Hex:	Hexadecimal (default)
Decimal	Decimal:	Signed decimal
Unsigned	Unsigned:	Unsigned decimal
Natural	Natural:	Common representation for the data type (e.g. decimal for int type values, hexadecimal for pointers)
Binary	Binary:	Binary
Octal	Octal:	Octal
Raw	Raw:	8-digit hexadecimal
Floating Point	Floating Point:	Floating point

#### Remove from Display

Deletes the register selected in the window from the window.

#### Add to Watch

Adds the register selected in the window to the [Watch Expressions] window.

#### Display All Registers

Displays the contents of all registers. The registers you removed using [Remove from Display] can be redisplayed.

### Displayed contents

The [Registers] window displays the contents of the CPU registers below. The register values displayed here are updated after program execution.

r0-r7, sp, pc, psr

## Operation

### Opening and closing the window

The [Registers] window opens automatically at debugger startup (unless the `gdbtk.ini` file is modified).

Clicking the [Close] button closes the window. The window can also be minimized.

To reopen the window, choose [Registers] from the [View] menu in the [Source] window or click the [Registers] button.



[Registers] button

Only one instance of the [Registers] window can be open at one time. Any attempt to open the [Registers] window while already open activates a new [Registers] window, which moves to the front of the screen.

### Changing display format

The display format of registers can be selected from the following:

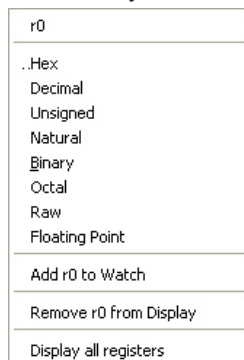
- Hex: Hexadecimal (default)
- Decimal: Signed decimal
- Unsigned: Unsigned decimal
- Natural: Common representation for the data type (e.g. decimal for `int` type values, hexadecimal for pointers)
- Binary: Binary
- Octal: Octal
- Raw: 8-digit hexadecimal
- Floating Point: Floating point

Choose the register whose display format you wish to change by clicking it. (Clicking once more deselects the register.) The name of a selected register is highlighted in blue.

**r0** 0x00006f Selected register

Next, choose the desired display format from the [Format] submenu of the [Register] menu.

Otherwise, you can choose from the popup menu shown below that appears by right-clicking.



### Rewriting register data

The contents of registers can be changed directly in the window. Choose the register whose content you wish to change by clicking it. Next, choose [Edit] from the [Register] menu. The register value is then highlighted in blue, with a text cursor displayed after the value, indicating that the register is in edit mode. A register can also be placed in edit mode by double-clicking it.

**r0** 0x00006f Edit mode

Enter a new value in decimal or hexadecimal to replace the existing value or alter the necessary digits of the existing value. You can use the arrow, backspace, and delete keys.

Note that if a value exceeding the register size is entered, only the 24 low-order bits are handled as valid, with those after bit 24 ignored.

**Hiding unnecessary registers**

You can choose not to display unnecessary registers.

Choose the register you wish to hide by clicking it. Next, choose [Remove from Display] from the [Register] menu or the popup menu that appears by right-clicking. The register is then hidden from the screen.

To redisplay hidden registers, choose [Display All Registers] from the [Register] menu or the popup menu that appears by right-clicking. All registers are then displayed. Note that [Display All Registers] is disabled and you cannot choose unless there is a hidden register.

**Adding registers to the [Watch Expressions] window**

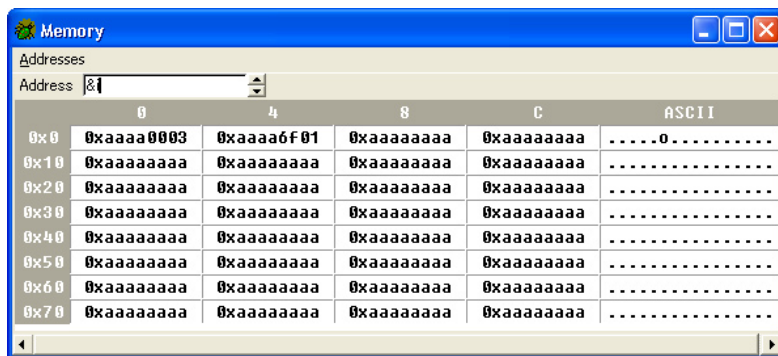
You can add a selected register to the watch list in the [Watch Expressions] window, so that you can monitor it while executing a program.

Choose the register you wish to add by clicking it. Next, choose [Add to Watch] from the [Register] menu or the popup menu that appears by right-clicking.

For details about the functions of the [Watch Expressions] window, see Section 10.4.7, "[Watch Expressions] Window".

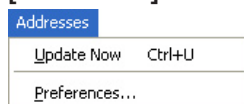
## 10.4.5 [Memory] Window

The [Memory] window is used to display and correct the contents of memory.



### Menus

#### [Addresses] menu



#### Update Now

Updates the displayed contents to the current state.

#### Preferences...

Displays the [Memory Preferences] dialog box in which you can choose the memory content display format and related settings. For details about the content set in this dialog box, see "Changing the basic settings" later in this section.

### Displayed contents

The [Memory] window displays the dumped data of a memory area. With default settings, 16 bytes (4 bytes × 4) of memory content, beginning with the start address of a selected line and the corresponding ASCII character, are displayed on one line.

The default data size and display format can be changed in the [Memory Preferences] dialog box. For details, see "Changing the basic settings" later in this section.

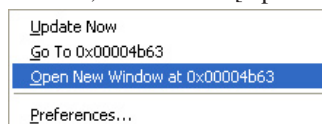
Note that 32-bit data are displayed in little endian format. In simulator mode, the memory content of any area is displayed in endian format (i.e., big or little endian as specified in a parameter file) set for that area.

### Operation

#### Opening and closing the window

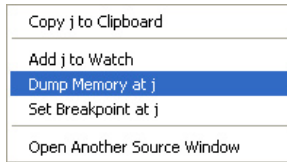
The [Memory] window opens automatically at debugger startup. Clicking the [Close] button closes the window. The window can also be minimized.

If necessary, multiple instances of the [Memory] window can be opened to display two or more areas simultaneously. To open another [Memory] window, select any item of data by clicking it in the current [Memory] window, then choose [Open New Window at <data>] from the popup menu that appears by right-clicking.



A new [Memory] window then appears, with contents displayed in it starting from the address of the selected data.

Otherwise, you can right-click a symbol name in the [Source] window and choose [Dump Memory at <symbol>] from the ensuing popup menu.



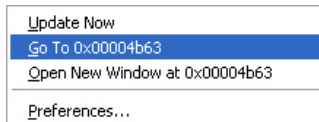
### Specifying the display start address

Enter the address from which you want the debugger to start displaying the contents of memory in the [Address] text box, then press the [Enter] key. The contents displayed in the window are then updated, with the specified address appearing at the upper-left corner of the window.

You can use a hexadecimal number (prefixed by 0x), decimal number, or global symbol to specify this address. If you specify with a symbol, the debugger uses the content of the symbol as an address. To specify an address at which a symbol is located, prefix the symbol with & to specify it.



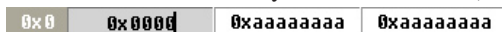
Otherwise, you can select data by clicking it in the [Memory] window, then choosing [Go to <data>] from the popup menu that appears by right-clicking. The selected data is then copied to the [Address] text box. Click in the [Address] text box and press the [Enter] key to have the debugger start display from that address.



The amount of data displayed in the window can be set to equal the window size (default) or a specified number of bytes in the [Memory Preferences] dialog box. If you specified a number of bytes and all data cannot be displayed within the current window size, a vertical scroll bar will appear. In this case, however, the displayed data cannot be scrolled exceeding the specified number of bytes. To display an area following or preceding the current area, click the [Scroll down] or [Scroll up] button provided at the side of the [Address] text box.

### Rewriting memory data

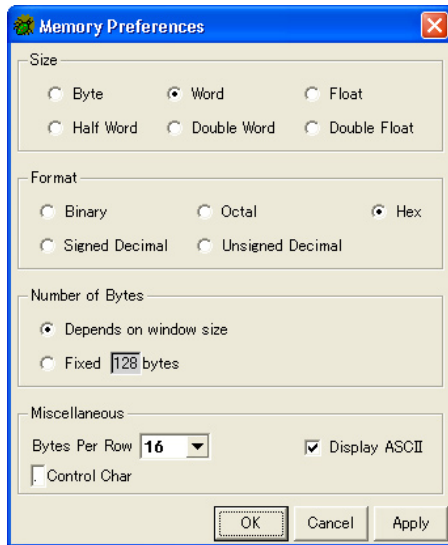
Choose a data cell you wish to change by clicking it in the window. Once a data cell is selected, the cell background is colored and a cursor appears after the value. Thus, you can enter hexadecimal or decimal data (or ASCII characters for entry in the ASCII field).



Press the [Enter] key to change the data and move the cursor to the next address, allowing you to enter data successively. However, you cannot use the backspace and delete keys. If you entered incorrect data, move the cursor back and reenter the data. Use the arrow keys to move between the cells.

## Changing the basic settings

The display format can be changed in the [Memory Preferences] dialog box that appears when you choose [Preferences...] from the [Addresses] menu or popup menu that appears by right-clicking in the window.



### Size

Use the radio buttons to specify the unit of data size you wish to display.

Byte:	Displayed in 1-byte units
Half Word:	Displayed in 2-byte units
Word:	Displayed in 4-byte units (default)
Double Word:	Displayed in 8-byte units
Float:	Single-precision floating-point
Double Float:	Double-precision floating-point

### Format

Use the radio buttons to select the format in which you wish to display data.

Binary:	Binary
Octal:	Octal
Hex:	Hexadecimal (default)
Signed Decimal:	Signed decimal
Unsigned Decimal:	Unsigned decimal

### Number of Bytes

Specify the amount of data to be displayed at one time.

**Depends on window size:** A number of lines equivalent to the window size is displayed. No vertical scroll bars are displayed. Use the [Scroll down] or [Scroll up] button to move the display area forward or backward.

**Fixed nnn bytes:** A specified number of bytes is displayed at one time. If the amount of data exceeds the window size, a vertical scroll bar is displayed. When you use the [Scroll down] or [Scroll up] button, the displayed data is scrolled the specified number of bytes each time you click.

For display in the horizontal direction of the window, the number of bytes displayed on one line is the same in either case. Note that exceeding the window size displays a horizontal scroll bar.

### Miscellaneous

**Bytes Per Row:** Set the number of bytes to be displayed on one line. The default is 16 bytes. Moreover, you can choose 4, 8, 32, 64, or 128 bytes.

**Display ASCII:** An ASCII character in an amount corresponding to one line of data is displayed at the end of each line (default setting). No ASCII characters are displayed when this check box is deselected.

**Control Char:** Specify the character to use for displaying control characters (when displaying ASCII characters). The default is a period.

### [Apply] button

Changes settings made in the dialog box, with the dialog box left open.

### [OK] button

Changes settings made in the dialog box, and closes the dialog box.

### [Cancel] button

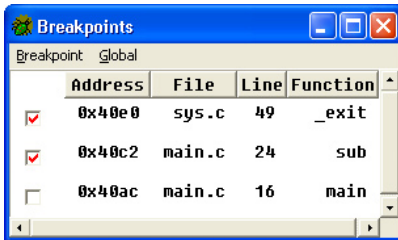
Closes the dialog box without changing the settings you made in it. (Settings already changed by [Apply] cannot be undone.)

Clicking [Apply] or closing this dialog box updates the contents displayed in the [Source] and other windows.



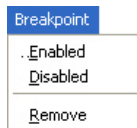
## 10.4.6 [Breakpoints] Window

The [Breakpoints] window is used to manage normal software, temporary software and hardware PC breakpoints.



### Menus

#### [Breakpoint] menu



##### Enabled

This menu command preceded by "✓" enables the breakpoint currently selected in the window (with the program being executed able to break at this point). After you choose a disabled breakpoint in the window, this menu command enables the selected breakpoint again.

##### Disabled

This menu command preceded by "✓" disables the breakpoint currently selected in the window (with the breakpoint ignored during program execution). After you choose an enabled breakpoint in the window, this menu command disables the selected breakpoint.

##### Remove

After you choose a breakpoint in the window, this menu command deletes the selected breakpoint.

#### [Global] menu



##### Disable All

Disables all PC breakpoints currently set.

##### Enable All

Enables all PC breakpoints currently set.

##### Remove All

Deletes all PC breakpoints currently set.

### Contents displayed

The following explains the contents displayed on each line of the breakpoint list.

#### Check box

Shows whether the breakpoint is enabled or disabled. When flagged by a check mark, the breakpoint is enabled; otherwise, the breakpoint is disabled. Moreover, the color of the check mark indicates whether the breakpoint is a normal software PC breakpoint (red by default), temporary software PC breakpoint (orange by default) or hardware PC breakpoint (blue by default). These check mark colors can be changed in the [Source Preferences] dialog box in the same way as for breakpoint marks in the [Source] window. (See Section 10.4.2, "[Source] Window".)

#### Address

Shows the address at which a PC breakpoint is set. Double-click on a line of the breakpoint list to highlight the line corresponding to that address in the [Source] window with the background color for browse. When this address is reached, the program is made to break (stop running) before executing the instruction stored at that location.

**File**

Shows the source file name corresponding to the break address.

**Line**

Shows a line number in the source file.

**Function**

Shows the function name corresponding to the break address.

**Operation****Opening/closing the window**

The [Breakpoints] window does not open automatically at debugger startup (unless the `gdbtk.ini` file is changed). To open the window, choose [Breakpoints] from the [View] menu in the [Source] window or click the [Breakpoints] button.



[Breakpoints] button

Click the [Close] button to close the window. The window can also be minimized.

Only one instance of the [Breakpoints] window can be open at a time. Any attempt to open the [Breakpoints] window while already open reactivates the [Breakpoints] window and moves it to the front of the screen.

**Choosing to enable or disable a breakpoint**

Click the check box at the beginning of a line in the breakpoint list to select or deselect it. Selecting the check box enables the breakpoint, so that the program being executed can be made to break at this point. Deselecting the check box disables the breakpoint, so that the breakpoint is ignored during program execution.

Moreover, you can select a breakpoint (highlighted with a blue background) by clicking it in the breakpoint list, then choose [Enabled] to enable the breakpoint or [Disabled] to disable the breakpoint from the [Breakpoint] menu or the popup menu that appears by right-clicking.



To enable or disable all breakpoints, choose [Disable All] or [Enable All] from the [Global] menu or the [Global] popup menu that appears by right-clicking.

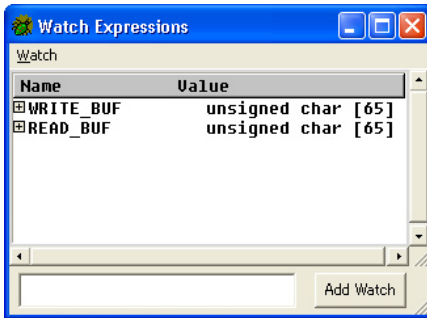
**Note:** Software PC breakpoints (including temporary breaks) and hardware PC breakpoints can be set at up to 200 locations and one location, respectively. The number of breakpoints set is not changed even when you disable one or more breakpoints here. When breakpoints are already set at 200 or one locations, no more breakpoints can be set regardless of whether you disable several breakpoints. To set additional breakpoints, you must first delete the unnecessary breakpoints.

**Deleting breakpoints**

Select a breakpoint you wish to delete by clicking on its line in the breakpoint list and choose [Remove] from the [Breakpoint] menu or popup menu that appears by right-clicking. To delete all PC breakpoints, choose [Remove All] from the [Global] menu or [Global] popup menu that appears by right-clicking.

## 10.4.7 [Watch Expressions] Window

The [Watch Expressions] window is used to monitor the values of global symbols and registers.



### Menus

#### [Watch] menu

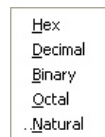


#### Edit

Places a symbol selected in the window into edit mode. While in this state, you can change the value of that symbol.

#### Format

Selects the format in which you wish to display the symbol value selected in the window.



Hex: Hexadecimal

Decimal: Signed decimal

Binary: Binary

Octal: Octal

Natural: Common representation for the data type (e.g. decimal for `int` type values, hexadecimal for pointers) (default)

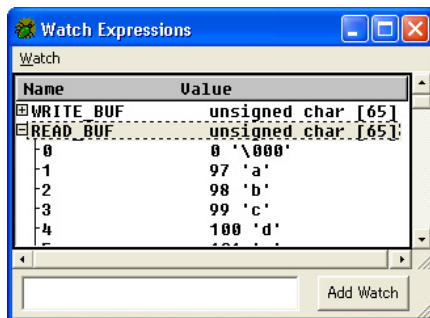
#### Remove

Deletes a symbol selected in the window from the watch list.

### Contents displayed

The names and values of symbols registered in the watch list are displayed.

Changing any symbol value during program execution updates the corresponding symbol value displayed in the window. Symbol values are not updated when changed by a command, etc.



When the symbol is an array or pointer, a [+] sign is displayed before the symbol name. Clicking the symbol changes the sign to [-] and displays information in the array or content of the address indicated by the pointer.

### Operation

#### Opening/closing the window

The [Watch Expressions] window does not open automatically at debugger startup (unless the `gdbtk.ini` file is changed). To open the window, choose [Watch Expressions] from the [View] menu in the [Source] window or click the [Watch Expressions] button.



[Watch Expressions] button

Click the [Close] button to close the window. The window can also be minimized. Only one instance of the [Watch Expressions] window can be open at a time. Any attempt to open the [Watch Expressions] window while already open reactivates the [Watch Expressions] window and moves it to the front of the screen.

### Registering symbols to be displayed

To display the contents of symbols in the [Watch Expressions] window, you must first register the symbols. Use one of the following methods to register symbols:

1. [Add Watch] button in [Watch Expressions] window

Enter a symbol name or a CPU register name beginning with \$ in the text box of the [Watch Expressions] window, then click the [Add Watch] button.



2. Symbol registration in [Source] window

To use this function, choose display format [SOURCE].

Place the cursor over a global variable name and right-click. Then choose [Add <symbol> to Watch] from the ensuing popup menu. (See Section 10.4.2, "[Source] Window".)

3. Register registration in [Registers] window

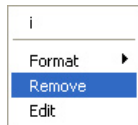
Select a register you wish to add by clicking it in the [Registers] window, then choose [Add to Watch] from the [Register] menu or the popup menu that appears by right-clicking. (See Section 10.4.4, "[Registers] Window".)

### Deleting registered symbols

Delete any symbols that are no longer necessary to watch from the window by following the method described below.

Select a symbol you wish to delete by clicking it.

Next, choose [Remove] from the [Watch] menu or the popup menu that appears by right-clicking.



### Changing display format

The format in which to display symbol values can be selected from the following:

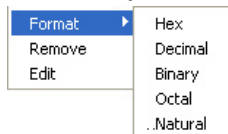
- Hex: Hexadecimal
- Decimal: Signed decimal
- Binary: Binary
- Octal: Octal
- Natural: Common representation for the data type (e.g. decimal for `int` type values, hexadecimal for pointers) (default)

Select a symbol whose display format you wish to change by clicking on its line in the window.

**i** ..... -1431655766 Selected symbol

Next, choose the desired display format from the [Format] submenu of the [Watch] menu.

Otherwise, you can choose a display format from the popup menu shown below that appears by right-clicking.



### Rewriting symbol values

Symbol values can be changed directly in the window. Select a symbol whose content you wish to change by clicking it in the window. Next, choose [Edit] from the [Watch] menu or the popup menu that appears by right-clicking. The symbol value will be highlighted with a blue background while a text cursor appears behind the value, indicating that the symbol is in edit mode. You also can enter edit mode by double-clicking a symbol value.

**i** ..... -1431655766 Edit mode

Enter a new value in decimal or hexadecimal to replace the existing value or change the necessary digits of the existing value. You can use the arrow, backspace, and delete keys.

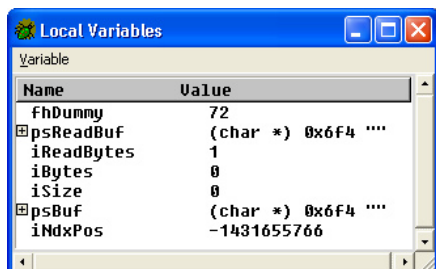
Note that entering a value exceeding the data size of the symbol only applies the low-order effective bits and ignores the bits outside the valid range.

## Limitations

- The symbol values displayed in the [Watch Expressions] window are not updated even when using a command to correct the contents of the values. Symbol values are only updated when changed during program execution.
- Only global variables and CPU register names can be registered in the [Watch Expressions] window. Use the [Local Variables] window to confirm the values of local variables.
- Do not edit the Windows screen property to change display colors during the debugger starting up as the debugger may hang up.
- The value of the variable, which is allocated to a register, may not be displayed properly due to the optimization by the C compiler.

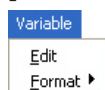
## 10.4.8 [Local Variables] Window

The [Local Variables] window is used to monitor the values of local variables.



### Menus

#### [Variable] menu

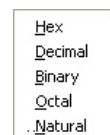


#### Edit

Places a variable selected in the window into edit mode. While in this state, you can change the value of that variable.

#### Format

Choose the format in which you wish to display the variable value selected in the window.



Hex: Hexadecimal

Decimal: Signed decimal

Binary: Binary

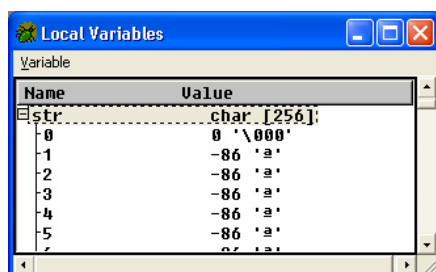
Octal: Octal

Natural: Common representation for the data type (e.g. decimal for `int` type values, hexadecimal for pointers) (default)

### Contents displayed

The local variable names and values defined in a function that includes the current PC address are displayed.

The variable values are updated after executing the program. When control is transferred to another function, the local variables displayed in the window are automatically changed.



If the variable is an array or pointer, a [+] sign is displayed before the variable name. Clicking the variable changes the sign to [-] and displays the information in the array or content of the address indicated by the pointer.

### Operation

#### Opening/closing the window

The [Local Variables] window does not open automatically at debugger startup (unless the `gdbtk.ini` file is changed).

To open the window, choose [Local Variables] from the [View] menu in the [Source] window or click the [Local Variables] button.



[Local Variables] button

Click the [Close] button to close the window. The window can also be minimized.

Only one instance of the [Local Variables] window can be open at a time. Any attempt to open the [Local Variables] window while already open reactivates the [Local Variables] window and moves it to the front of the screen.

## Changing display format

The format in which to display variable values can be selected from the following:

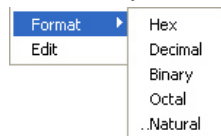
- Hex: Hexadecimal
- Decimal: Signed decimal
- Binary: Binary
- Octal: Octal
- Natural: Common representation for the data type (e.g. decimal for `int` type values, hexadecimal for pointers) (default)

Select a variable whose display format you wish to change by clicking on its line in the window.

**i** Selected variable

Next, choose the desired display format from the [Format] submenu of the [Variable] menu.

Otherwise, you can choose a display format from the popup menu shown below that appears by right-clicking.



## Rewriting variable values

Variable values can be changed directly in the window.

Select a variable whose content you wish to change by clicking it in the window.

Next, choose [Edit] from the [Variable] menu or the popup menu that appears by right-clicking. The variable value will be highlighted with a blue background while a text cursor appears behind the value, indicating that the variable is in edit mode. You also can enter edit mode by double-clicking a variable value.

**i** **2048** Edit mode

Enter a new value in decimal or hexadecimal to replace the existing value or change the necessary digits of the existing value. You can use the arrow, backspace, and delete keys.

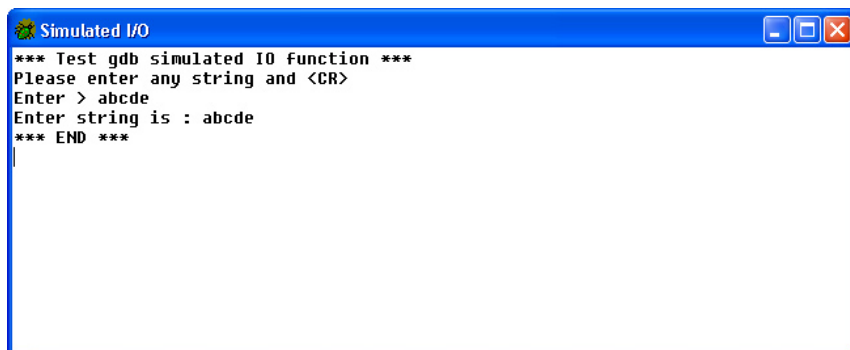
Entering a value exceeding the data size of the variable only applies the low-order effective bits and ignores the bits outside the valid range.

## Limitations

- The local variable values displayed in the [Local Variables] window are not updated even when using a command to correct the contents of these values. Variable values are only updated when changed during program execution.
- The value of the variable, which is allocated to a register, may not be displayed properly due to the optimization by the C compiler.

## 10.4.9 [Simulated I/O] Window

The [Simulated I/O] window is used for input/output by the simulated I/O function.



```

*** Test gdb simulated IO function ***
Please enter any string and <CR>
Enter > abcde
Enter string is : abcde
*** END ***

```

### Contents displayed

The contents supplied from stdin and those output to stdout by the simulated I/O function are displayed.

### Operation

#### Opening/closing the window

The [Simulated I/O] window does not open automatically at debugger startup (unless the `gdbtk.ini` file is changed). To open the window, choose [SimI/O] from the [View] menu in the [Source] window.

In the following cases, the [Simulated I/O] window opens automatically:

- When the data input source is set in the window by the `c17 stdin` command and the program is made to break at a specified breakpoint
- When the data output destination is set in the window by the `c17 stdout` command and the program is made to break at a specified breakpoint

Click the [Close] button to close the window. The window can also be minimized.

Only one instance of the [Simulated I/O] window can be open at a time. Any attempt to open the [Simulated I/O] window while already open reactivates the [Simulated I/O] window and moves it to the front of the screen.

#### Entering data

The data to be supplied from stdin can be entered from this window.

For details, see Section 10.6.8, "Simulated I/O".

### Limitations

Up to 256 lines can be displayed in the [Simulated I/O] window.



## 10.4.10 [Trace] Window

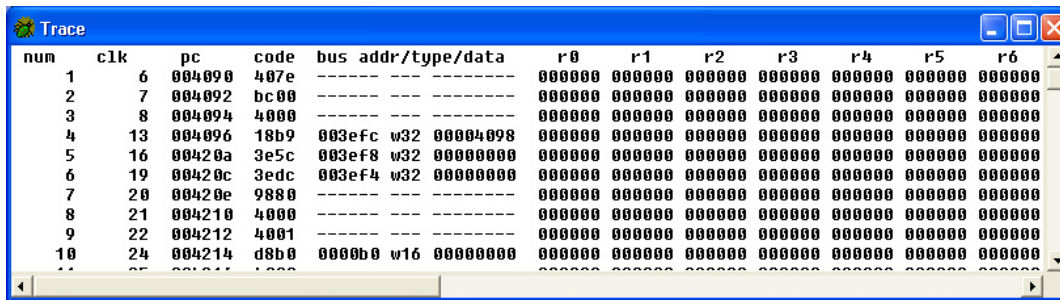
The [Trace] window is used to display trace data.

### Contents displayed

The [Trace] window displays trace data that indicates the result of each instruction executed.

For details about trace mode and trace content, see Section 10.6.7, "Trace Functions".

The trace function is available only in simulator mode.



num	clk	pc	code	bus	addr/type/data	r0	r1	r2	r3	r4	r5	r6
1	6	004090	407e	-----	---	000000	000000	000000	000000	000000	000000	000000
2	7	004092	bc00	-----	---	000000	000000	000000	000000	000000	000000	000000
3	8	004094	4000	-----	---	000000	000000	000000	000000	000000	000000	000000
4	13	004096	18b9	003efc	w32 00004098	000000	000000	000000	000000	000000	000000	000000
5	16	00420a	3e5c	003ef8	w32 00000000	000000	000000	000000	000000	000000	000000	000000
6	19	00420c	3edc	003ef4	w32 00000000	000000	000000	000000	000000	000000	000000	000000
7	20	00420e	9880	-----	---	000000	000000	000000	000000	000000	000000	000000
8	21	004210	4000	-----	---	000000	000000	000000	000000	000000	000000	000000
9	22	004212	4001	-----	---	000000	000000	000000	000000	000000	000000	000000
10	24	004214	d8b0	0000b0	w16 00000000	000000	000000	000000	000000	000000	000000	000000

In simulator mode, once the trace function is turned on (by the `c17 tm` command), all program execution from that time on is displayed in the trace window (except when selecting file output).

The following contents are displayed:

- Number of instructions executed
- Executed address, code, and disassembled content
- Content of memory access (address, R/W, and data)
- Source code
- Register content

### Operation

#### Opening/closing the window

The [Trace] window does not open automatically at debugger startup (unless the `gdbtk.ini` file is changed). To open the window, choose [Trace] from the [View] menu in the [Source] window.

Click the [Close] button to close the window. The window can also be minimized.

Only one instance of the [Trace] window can be open at a time. Any attempt to open the [Trace] window while already open reactivates the [Trace] window and moves it to the front of the screen.

#### Updating display

When the trace function is turned on (by the `c17 tm` command), trace results are displayed successively along with the progress of program execution. When program execution stops, so does display.

When the trace function is turned off (default), display is not updated even by executing the program.

### Limitations

Up to 255 lines can be scrolled in the [Trace] window.

## 10.5 Method of Executing Commands

Executing debugging commands can perform all debug functions. This section describes the method of executing these commands. For command parameters and other details, see the explanation of each command described later in this manual.

### 10.5.1 Entering Commands From the Keyboard

Use the [Console] window to enter commands. When the [Console] window is behind other windows, click somewhere in the [Console] window to activate it. If the [Console] window is not yet displayed, choose [Console] from the [View] menu in the [Source] window or click the [Console] button.



[Console] button

When the [Console] window has a prompt "(gdb)" appearing at the last line, with a cursor blinking behind the prompt, you are ready to enter a command.

Therefore, enter a debugging command in lowercase letters.

#### General command input format

```
(gdb) command [parameter [parameter ... parameter]]
```

A space is required between the command and a parameter, and between parameters.

If you have entered an incorrect command by mistake, use the arrow (←, →), [Backspace], or [Delete] keys to correct it.

When you have finished entering a command, press the [Enter] key to execute the command.

Example: (gdb) continue (entry of command only)

(gdb) target icd usb (entry of command and parameters)

#### Successive execution by the [Enter] key

Once a command is executed, you can repeat the same operation or display memory contents following the previously displayed contents (x command) by simply using the [Enter] key. This function is effective until you execute another command.

When a source command is used to execute a command file, all commands written in the command file are executed. This function may be used to execute multiple commands in succession.

#### Command history

In the [Console] window, you can use the arrow keys (↑, ↓) to redisplay commands that have already been executed at the prompt position, one command at a time. Thus, clicking the [Enter] key reexecutes the command currently displayed.

## 10.5.2 Parameter Input Format

### Numeric input

Parameters used to specify an address or data in a command must be entered in decimal (by default). To enter a parameter in hexadecimal, add 0x (or 0X) to the beginning of the value. Only characters 0 to 9, 'a' to 'f' and 'A' to 'F' are recognized as hexadecimal.

To specify an immediate address in a command that causes the program to break, add \* to the beginning of the value, as shown below.

Example: (gdb) break \*0xc00040

You need not add this asterisk for address parameters not preceded by \* in the explanation of each command format.

### Specifying a source line number

For commands that cause the program to break, you can specify a breakpoint by source line number. However, this is limited to only when debugging an elf format object file that includes information on source line numbers.

To specify a line number, use the format shown below.

*Filename:LineNo.*

*Filename:* Source file name

*Filename:* can be omitted when specifying a line number existing in the current file (one that includes code for the current PC).

*LineNo.:* Line number

Line numbers can only be specified in decimal.

Example: main.c:100

### Address specification by a symbol

You can use a symbol to specify an address. However, this is limited to only when debugging an elf format object file that includes symbol information.

### Entering a file name

For file names in other than the current directory, always be sure to specify a path.

Only characters 'a' to 'z,' 'A' to 'Z,' 0 to 9, /, and \_ can be used.


Drive names must be specified in /cygdrive/<drive name>/ format, with / instead of \ used for delimiting the path.

Example: (gdb) file /cygdrive/c/EPSON/gnu17/sample/txt/sample.elf

### 10.5.3 Using Menus and Toolbar To Execute Commands

Some commands are registered in the [File], [Reset], and [Control] menus, and on the toolbar in the [Source] window as shown in Section 10.4.2. These commands can be executed by selecting one from a menu or clicking the relevant toolbar button. Moreover, each window implements an equivalent function to execute a command. Table 10.5.3.1 below lists the registered commands.

Table 10.5.3.1 Commands specifiable from menus, toolbar, and windows

Command	Window	Menu/other	Button
continue	[Source]	[Control]-[Continue]	
until	[Source]	Popup menu	-
step	[Source]	[Control]-[Step]	
stepi	[Source]	[Control]-[Step Asm Inst]	
next	[Source]	[Control]-[Next]	
nexti	[Source]	[Control]-[Next Asm Inst]	
finish	[Source]	[Control]-[Finish]	
User command *	[Source]	[User]-[User Command]	
c17 rst *	[Source]	[Reset]-[Reset]	
break	[Source]	Popup menu, Mouse button	-
tbreak	[Source]	Popup menu	-
disable	[Breakpoints]	[Breakpoint]-[Disabled], Popup menu	-
enable	[Breakpoints]	[Breakpoint]-[Enabled], Popup menu	-
delete	[Breakpoints]	[Breakpoint]-[Remove], Popup menu	-
clear	[Source]	Mouse button	-
x /b, x /h, x /w	[Memory]	Address entered from keyboard	-
set {char}, set {short}, set {long}	[Memory]	Data entered from keyboard	-
info reg	[Registers]	-	-
set \$Register	[Registers]	Data entered from keyboard	-
info locals	[Local Variables]	-	-
print	[Watch Expressions]	Symbol name entered from keyboard	-
file	[Source]	[File]-[Open...]	-
source	[Source]	[File]-[Source...]	-
quit	[Source]	[File]-[Exit]	[Close]

\* The menu command/button executes the predetermined command file using the `source` command. The contents of the command file may be freely edited by the user.

However, do not write a command that executes the command file itself, for example, when "`source userdefine.gdb`" is written in the `userdefine.gdb` file, as it will enter an endless loop.

Furthermore, the file name and directory cannot be changed. If the command file does not exist in the directory in which the file was installed, an error occurs when it is executed by the menu command or the button.

**[User]-[User Command]**

Executes the command file `\gnu17\userdefine.gdb`.

Contents of `userdefine.gdb` at shipment

```
#Edit user command
#c17 rst                (No command executed)
```

The debugger uses the image file shown below to display this button. This button can be customized by replacing the contents in this file with a user created 24 × 24-dot button image.

`\gnu17\utility\share\gdbtcl\images2\userdefine.gif`

**[Reset]-[Reset]**

Executes the command file `\gnu17\reset.gdb` that can be edited by the user.

Contents of `reset.gdb` at shipment

```
c17 rst                (Resets the CPU)
```

**Note:** Compared to actual command input, this method is subject to some limitations (e.g., parameters cannot be specified).

The commands executed from menus, the toolbar, or windows are not displayed in the prompt part of the [Console] window.

## 10.5.4 Using a Command File To Execute Commands

You can use a command file to execute a series of debugging commands written in the file.

### Creating a command file

Create a command file as a text file using a general-purpose editor, etc.

### Example of a command file

Only one command can be written per line.

Example:

<code>c17 rpf c17.par</code>	Sets memory map information.
<code>file sample.elf</code>	Loads debug information.
<code>target sim</code>	Connects the target.
<code>load</code>	Loads a program.
<code>c17 rst</code>	Resets the CPU.
<code>c17 stdout 1 WRITE_FLASH WRITE_BUF stdout.txt</code>	Sets stdout.
<code>c17 stdin 1 READ_FLASH READ_BUF stdin.txt</code>	Sets stdin.
<code>break _exit</code>	Sets a software PC breakpoint.
<code>cont</code>	Executes the program.
<code>c17 stdout 2</code>	Clears stdout.
<code>c17 stdin 2</code>	Clears stdin.

### Loading/executing a command file

There are two methods of loading and executing a command file:

#### 1. Execution by a startup option

By specifying the `-x` option (or `--command` option) in the debugger startup command, you can execute one command file at debugger startup.

Example: `c:\EPSON\gnu17\gdb -x startup.cmd`

#### 2. Execution by a command

A command named "source" is available to execute a command file. The `source` command loads a specified file and executes the commands in it in the order written.

Example: `(gdb) source startup.cmd`

The commands written in a command file are displayed in the [Console] window.

The `source` command can also be executed from the [File] menu in the [Source] window.

### Executing a command file repeatedly

Once a command file is executed using the `source` command, you can execute it repeatedly by simply clicking the [Enter] key thereafter. In this case, all commands written in the command file are executed. This function is effective until you execute another command.

### Command execution intervals

When you enter the `--c17_cmw` option, a wait time specified in seconds is inserted between each command. The wait time can be specified from 1 to 256 seconds. If any other value is specified, a 1-second wait time is assumed. When the debugger is started without specifying the `--c17_cmw` option, no wait time is inserted between each command.

## 10.5.5 Log Files

The commands executed and execution results can be saved as a log file in text format. Log files enable you to confirm the debugging procedure and contents at a later time.

### Example command

```
(gdb) c17 log test.log  Starts logging.
      :                  Log mode
(gdb) c17 log           Finishes logging.
```

After logging is started by the `c17 log` command, the debugger saves a log until the next time you execute this command.

### Saved contents of a log

All commands executed and execution results are saved. This includes commands that have been executed from menus or toolbar buttons and are not displayed in the [Console] window.

## 10.6 Debugging Functions

This section outlines the debugging functions of **gdb**, separately for each function. For details about each debugging command, see Section 10.7, "Command Reference".

### 10.6.1 Connect Modes

Note that **gdb** supports two connect modes, of which the mode used is set by the **target** command.

#### ICD Mini mode

In this mode, the ICD Mini (S5U1C17001H) or ICD board is used to perform debugging. The program is executed on the target board.

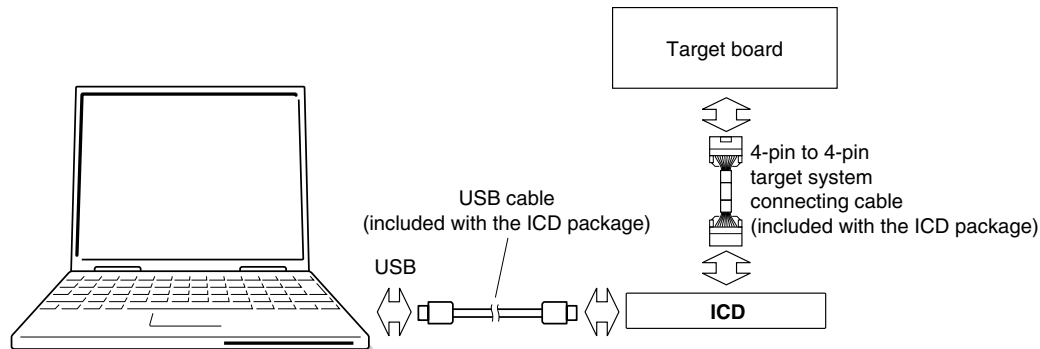


Figure 10.6.1.1 Example of debugging system using an ICD

#### Specification method

Command: (gdb) **target icd usb**

Specification in **IDE**:

Select "ICD Mini" from the [Debugger:] combo box in the [Create a simple startup command] dialog box to generate a startup command file.

To start in ICD Mini mode, make sure an ICD and target board are connected correctly, and that the power for these units is turned on. For details on how to use the ICD, refer to the manual for the ICD used.

Note that the trace mode is not available in ICD Mini mode.

#### Simulator (SIM) mode

In simulator mode, target program execution is simulated in internal memory of a personal computer, with no other tools required. However, the ICD-dependent functions cannot be used in this mode.

#### Specification method

Command: (gdb) **target sim**

Specification in **IDE**:

Select "Simulator" from the [Debugger:] combo box in the [Create a simple startup command] dialog box to generate a startup command file.

The trace mode is available in simulator mode. The flash writer function cannot be used.

## 10.6.2 Loading a File

### Types of files

The debugger **gdb** can load an elf format object file to debug.

### File loading procedure

Use the following two commands to load a file:

**file** command: Loads debugging information.

**load** command: Loads object code into the target.

Aside from the above, the debugger is provided with the **c17 rpf** command, which can be used to load a parameter file to set memory map information of the target.

The **file** command must be executed before the **target** or **load** command. The **c17 rpf** command must also be executed before the **file** command.

The following shows the basic procedure to execute a series of operations from loading a file to debugging.

```
(gdb) c17 rpf sample.par (Sets map information.)
(gdb) file sample.elf    (Loads debugging information.)
(gdb) target icd usb     (Connects the target.)
(gdb) load               (Loads the program.)
(gdb) c17 rst            (Resets the CPU.)
```

To debug a program written in target ROM, there is no need to execute the **load** command. In this case, the **file** command can also be used to load debugging information for source-level debugging.

### Notes

The **load** command only loads several areas (containing the code and data) of an object file. All other areas are left intact in the original state before the **load** command was executed.

The debugger **gdb** loads source files according to debugging information to display the sources. Therefore, both contents and storage locations (directories) of the source files must be in the same state as at elf object file creation.



### 10.6.3 Source-Level Debugging Function

When **gdb** is up and running, you can debug a program while displaying the C source and assembly source in the [Source] window.

The screenshot shows a window titled "Source Window" with a menu bar (File, User, Reset, View, Control, Preferences, Help) and a toolbar. The toolbar includes icons for stopping, stepping, and other debugging actions. The main area displays the source code for "main.c" in the "sub" function. The code is as follows:

```

1 /* C main program */
2
3 #include "stdio.h"
4
5 int i;
6
7 void sub( int k );
8
9 main()
10 {
11     int j;
12     i = 0;
13
14     for( j = 0; j < 6; ++j )
15     {
16         sub(j);
17     }
18 }
19
20 void sub( int k)
21 {
22     if (k & 0x1)
23     {
24         i++;
25         putc( 'o',stdout );
26     }

```

The status bar at the bottom of the window reads "Program stopped at line 24".

For details about the source display, see Section 10.4.2, "[Source] Window".

## 10.6.4 Manipulating Memory, Variables, and Registers

The debugger **gdb** can perform operations in memory and registers. 16-bit and 32-bit data are accessed and displayed in little endian format. In simulator mode, however, a specific external memory area can be displayed in big endian format (as set in a parameter file).

### Manipulating memory areas

Following operations can be performed in memory areas. You can use such symbols as variable names to specify addresses. Any operation described below can be processed in units of bytes, 16 bits, or 32 bits.

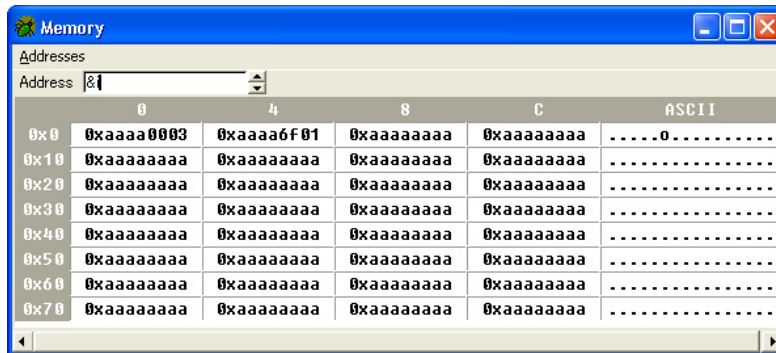
#### Memory dump (**x /b**, **x /h**, **x /w** commands)

Dumps memory contents for a specified amount of data from a specified address for display on the screen.

Example: Memory dump for 16-bit  $\times 16$  from `_START_text`

```
(gdb) x /16h _START_text
0xc00000 <_START_text>: 0x0004 0x00c0 0xc020 0x6c0f 0xa0f1 0xc000 0xc000 0x6c0f
0xc00010 <boot+12>:      0xc000 0xc000 0x1c04 0xdfdf 0xdfdf 0x1ef5 0x0200 0x6c04
```

Memory data can be displayed in the [Memory] window. For display in the [Memory] window and how to operate in it, see Section 10.4.5, "[Memory] Window".



#### Entering data (**set {char}**, **set {short}**, **set {long}** commands)

Writes specified data to a specified address.

Data can also be entered or changed in the [Memory] window.

Example: Setting `int i` to `0x5555`

```
(gdb) set {short}&i=0x5555
```

#### Rewriting a specified area (**c17 fb**, **c17 fh**, **c17 fw** commands)

Rewrites all of a specified area with specified data.

Example: Writing `0x00000001` (32 bits)  $\times 4$  to addresses `0x0` through `0xf`

```
(gdb) c17 fw 0x0 0xf 0x1
```

Start address = `0x0`, End address = `0xc`, Fill data = `0x1` .....done

#### Copying a specified area (**c17 mvb**, **c17 mvh**, **c17 mvw** commands)

Copies the content of a specified address to another area.

Example: Copying 8 bytes from addresses `0x0` through `0x7` to an 8-byte area beginning with address `0x8`

```
(gdb) c17 mvb 0x0 0x7 0x8
```

Start address = `0x0`, End address = `0x7`, Destination address = `0x8` .....done

#### Saving memory contents (**c17 df** command)

Outputs the contents in a specified range of memory to a file in binary, text, or Motorola S3 format.

Example: Saving the contents of addresses `0x80000` to `0x80103` as a Motorola S3 format file named `dump.mot`

```
(gdb) c17 df 0x80000 0x80103 3 dump.mot
```

Start address = `0x80000`, End address = `0x8011f`, File type = Motorola-S3

Processing 00080000-0008011F address.

**Specification of target memory read mode (c17 readmd command) ICD Mini mode only**

In an ordinary memory read by the `x` command or `c17 df` command, data is always read out in units of bytes, regardless of accessed data size. If this read method becomes inconvenient, use the `c17 readmd` command to alter the read method so that memory data will be read out from the correct boundary address in specified units. Note that doing so will increase memory dump time.

Example: The read method is modified so that memory data will be read out from the correct boundary address in specified units.

```
(gdb) c17 readmd 1
```

**Variable list**

A list of variables can be displayed.

**Displaying global variables (info var, print commands)**

You can display a list of global variables, static variables, or section symbols by using the `info var` command. You also can display the contents of variables by using the `x` or `print` command.

Example: Displaying a list of global symbols

```
(gdb) info var
```

All defined variables:

File main.c:

```
int i;
```

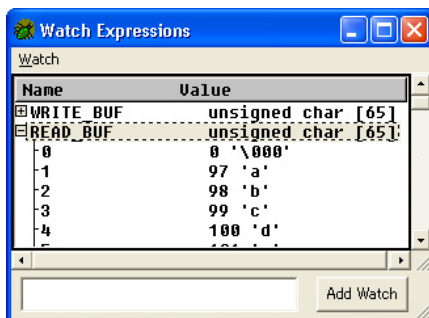
Non-debugging symbols:

```
0x00000000 __START_bss
```

```
0x00000004 __END_bss
```

```
0x00000004 __END_data
```

```
0x00000004 __START_data
```



Moreover, when the [Watch Expressions] window has global variables registered in it, you can monitor the values of those variables. For display in the [Watch Expressions] window and how to operate in it, see Section 10.4.7, "[Watch Expressions] Window".

**Displaying local variables (info locals command)**

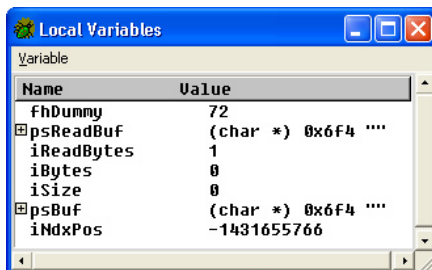
You can display a list of local variables and values defined in a function that includes the current PC address by using the `info locals` command.

Example: Displaying local variables defined in current function

```
(gdb) info locals
```

```
i = 0
```

```
j = 2
```



Moreover, by leaving the [Local Variables] window open, you can monitor the values of all local variables defined in the current function. For display in the [Local Variables] window and how to operate in it, see Section 10.4.8, "[Local Variables] Window".

## Register operation

The following operations can be performed in registers.

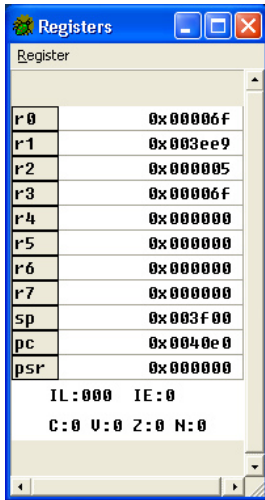
### Displaying registers (`info reg` command)

You can display the contents of all CPU registers or the content of a specified register in the [Console] window.

Example: Displaying the values of all registers

```
(gdb) info reg
r0          0xd20          3360
r1          0xaaaaaaaa      11184810
r2          0xaaaaaaaa      11184810
r3          0xaaaaaaaa      11184810
r4          0x690          1680
r5          0xaaaaaaaa      11184810
r6          0x0            0
r7          0xaaaaaaaa      11184810
sp          0x7f8          2040
pc          0x4090         16528
psr         0x0            0
```

The [Registers] window also displays register values. For display in the [Registers] window and how to operate in it, see Section 10.4.4, "[Registers] Window".



### Altering register values (`set $` command)

You can set the contents of CPU registers to any desired values.

Example: Setting the r1 register to 0x10000

```
(gdb) set $r1=0x10000
```

The register values can also be rewritten in the [Registers] window. (See Section 10.4.4, "[Registers] Window".)

## 10.6.5 Executing the Program

The debugger can execute the target program continuously or one step at a time (single-stepping).

### Continuous execution

#### Continuous execution commands (continue, until commands)

The continuous execution commands execute the loaded program continuously from the current PC address.

**continue** command: When executing the program continuously, you can disable the current breakpoint a specified number of times.

Example 1: Executing the program continuously from current PC

```
(gdb) cont
Continuing.
```

Example 2: Executing the program continuously from current PC after specifying that current breakpoint be skipped 4 times

```
(gdb) continue 5
Will ignore next 4 crossings of breakpoint 1. Continuing.
```

**until** command: You can specify a temporary PC breakpoint that is effective for only one break and cause the program to stop running at that position.

Example: Executing the program continuously from current PC to 10th line in `main.c` and causing the program to break immediately before executing 10th line in `main.c`

```
(gdb) until main.c:10
main () at main.c:10
```

The commands above can also be executed in the [Source] window.

To execute the `continue` command:

- Choose [Continue] from the [Control] menu.
- Click the [Continue] button.



\* You cannot specify the number of times that a break should be disabled.

To execute the `until` command:

- Choose [Jump to Here] from a popup menu.
- \* To display a popup menu, right-click at the beginning of the source line where you wish to set a temporary PC breakpoint.



For details on how to operate in the [Source] window, see Section 10.4.2, "[Source] Window."

#### Stopping continuous execution

The program being executed does not stop until made to break by one of the following causes:

- Break conditions set by a break setup command are met (including a temporary break specified by the `until` command).
- Forcible break (generated by clicking the [Stop] button)
- Other causes of break generated



\* If the program does not stop, it can be forcibly made to break by using this button.

When the program stops, the cause of break and halted position are displayed in the [Console] window. Moreover, the contents displayed in the [Source] and [Registers] windows are updated.

## Single-stepping a program

### Types of single-step commands

There are three types of single-step commands:

#### Single-stepping all codes (**step** and **stepi** commands)

The program is executed one step or a specified number of steps from the current PC address. When a function or subroutine call is encountered, lines or instructions in the called function or subroutine are single-stepped.

**step** command: The program is single-stepped one source line at a time.

Example: Single-stepping the program by one source line indicated by the current PC

```
(gdb) step
```

**stepi** command: The program is single-stepped one assembler instruction at a time.

Example: Single-stepping the program by ten instructions from the address indicated by the current PC

```
(gdb) stepi 10
main () at main.c:13
```

#### Single-stepping all codes except functions/subroutines (**next** and **nexti** commands)

The program is executed one step or a specified number of steps from the current PC address. When a function or subroutine call is encountered, all lines or instructions in the called function or subroutine are executed successively as one step. Otherwise, these commands operate the same way as the **step** and **stepi** commands.

**next** command: The program is single-stepped one source line at a time.

Example: Single-stepping the program by one source line indicated by the current PC, with any and all lines in a called function executed successively as one step

```
(gdb) next
```

**nexti** command: The program is single-stepped one assembler instruction at a time.

Example: Single-stepping the program by ten instructions from the address indicated by the current PC, with any and all lines in a called subroutine executed successively as one step

```
(gdb) nexti 10
main () at main.c:13
```

#### Terminating a function/subroutine (**finish** command)

When the program has been halted within a function/subroutine, this command single-steps the program until returning to the caller.

Example: Terminating the current function

```
(gdb) finish
Run till exit from #0 0x00c00040 in sub (k=1) at main.c:22
main () at main.c:14
Value returned is $1 = 480
```

When executing the commands above from the command prompt, you can specify the number of steps to execute, for up to 0x7fffffff.

When the program stops, the source at the halted position is displayed in the [Console] window. The contents displayed in the [Source] and [Registers] windows are also updated.

The commands above can also be executed using the menus or toolbar buttons in the [Source] window. However, you cannot specify the number of steps to execute (since only one step is always executed).

To execute the **step** command:

- Choose [Step] from the [Control] menu.
- Click the [Step] button.



[Step] button

To execute the **stepi** command:

- Choose [Step Asm Inst] from the [Control] menu.
- Click the [Step Asm Inst] button.

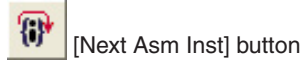


[Step Asm Inst] button

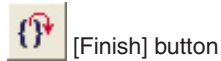
- To execute the `next` command:
- Choose [Next] from the [Control] menu.
  - Click the [Next] button.



- To execute the `next i` command:
- Choose [Next Asm Inst] from the [Control] menu.
  - Click the [Next Asm Inst] button.



- To execute the `finish` command:
- Choose [Finish] from the [Control] menu.
  - Click the [Finish] button.



### Breaking program execution during single-stepping

When the program is run after specifying the number of steps to execute, the program will be made to break before completion by one of the following causes:

- Forcible break (generated by clicking the [Stop] button)
- Other causes of break than those set by the user

During single-stepping, the program does not stop at PC breakpoints.



- \* If the program does not stop, it can be forcibly made to break by using this button.

When the program stops, the cause of break and halted position are displayed in the [Console] window.

### Calling a user function (`callmd` and `call` commands)

The `call` command can be used to call a user function.

The `callmd` command is used to set the destination (screen or file) to which execution results of the `call` command are to be output.

### HALT and SLEEP states and interrupts

The `halt` and `slp` instructions are always executed to place the CPU in standby mode, regardless of whether the program is executed continuously or single-stepped. The CPU exits standby mode when an external interrupt is generated. Clicking the [Stop] button also releases the CPU from standby mode.

## Measuring the execution cycles/execution time

In ICD Mini or simulator mode, you can measure the program execution cycles or time.

### Execution counter

The ICD contains one 31-bit execution counter. By using the `c17 clockmd` command, this counter can be set for measurement in units of execution time or the number of cycles executed. In simulator mode, the counter only counts the number of cycles executed.

The measurement results can be displayed in the [Console] window by using the `c17 clock` command.

If the counter exceeds the maximum measurable value, a message ("clock timer overflow") is displayed.

### Integrating mode and reset mode

With the debugger's default settings, the execution cycle counter is set to integrating mode. In this mode, the values measured by the counter each time are integrated until the counter is reset.

Reset mode can be set using the `c17 clockmd` command. In cases where the counter is set to reset mode, the counter is reset when the program is started by entering an execution command and continues counting until the program terminates (or made to break).

### Resetting the execution counter

The execution counter is reset in the following cases:

- When the mode of the execution counter is changed (from integrating mode to reset mode, or vice versa) by the `c17 clockmd` command
- When the program is started while the counter is set to reset mode
- When the CPU is reset
- When the `c17 timebrk`, `step`, `stepi`, `next`, `nexti` or `finish` command is executed in ICD Mini mode

## Resetting the CPU

The CPU is reset by the `c17 rst` command. This command can also be executed using the menu or toolbar button in the [Source] window.

To execute the `c17 rst` command:

- Choose [Reset] from the [Reset] menu.
- Click the [Reset] button.

(\gnu17\reset.gdb)



[Reset] button

**Note:** This menu command/button executes the predetermined command file. The command file at shipment contains the above reset command only (it may be edited by the user).

When the CPU is reset, its internal registers and other components are initialized as shown below.

(1) Internal registers of the CPU

```
r0-r7: 0x000000
pc:    Boot address (reset vector in the trap table)
sp:    0xffffc
psr:   0x00 (IL = 000, IE = 0, CVZN = 0000)
```

(2) The execution counter is cleared to 0.

(3) The [Source] and [Registers] windows reappear.

Because the PC is set to the boot address, the [Source] window redisplay the program beginning with that address. The [Registers] window reappears with the initialized values.

Memory contents are not changed.



## 10.6.6 Break Functions

The target program being executed is made to break by one of the following causes:

- Break conditions set by a break setup command are met.
- A forcible break is applied (by clicking the [Stop] button).
- An illegal attempt is made to access memory, etc.

### Command-actuated breaks

The debugger **gdb** supports the following two types of breaks for which break conditions can be set by a command:

1. Software PC break
2. Hardware PC break

In all cases, the program being executed is made to break when break conditions are met.

#### Software PC breaks (**break** and **tbreak** commands)

This type of break occurs when the executed PC address matches the address set by a command. The program is actually made to break before executing the instruction at that address. Breakpoints can be set at up to 200 address locations.

There are two types of software PC breaks: normal and temporary. Both are the same in terms of functionality. The only difference is that a normal software PC breakpoint remains effective until being cleared by a command, regardless of how many times the program is made to break (a hit). Conversely, a temporary software PC breakpoint is cleared after one break hit.

**break** command: This command sets a normal software PC breakpoint.

Example: To set a software PC breakpoint at address 0xc0001c

```
(gdb) break *0xc0001c
Breakpoint 1 at 0xc0001c: file main.c, line 7.
```

**tbreak** command: This command sets a temporary software PC breakpoint.

Example: To set a temporary software PC breakpoint at address 0xc0001e

```
(gdb) tbreak *0xc0001e
Breakpoint 2 at 0xc0001e: file main.c, line 10.
```

When a software PC break occurs, the debugger waits for command input after displaying the following message:

```
(gdb) continue
Continuing.
```

```
Breakpoint 1, main () at main.c:7
```

Breakpoints can be set by specifying a source line number or function/label name, as well as by directly specifying addresses. Specifying a source line number sets a breakpoint at the address of the first assembler instruction to be executed among those for which the specified line is expanded. Specifying a line that is not expanded to such assembler instructions, such as a variable declaration that does not involve initialization, a breakpoint will be set at the line that contains the first instruction to be executed next.

Example: To set a software PC breakpoint at line 7 in `main.c`

```
(gdb) break main.c:7
Breakpoint 1 at 0xc0001c: file main.c, line 7.
```

Specifying a function name sets a breakpoint at the source line that contains the first instruction to be executed in the function. No breakpoints are set at lines consisting only of variable declarations.

Example: To set a software PC breakpoint in function `main`

```
(gdb) break main
Breakpoint 1 at 0xc0001c: file main.c, line 7.
```

Although the `ld` instruction to save registers is added at the beginning of a function when compiling source files, the address of this instruction does not constitute a breakpoint because it does not correspond to any source line. However, it can be set as a breakpoint by specifying that address.

Software PC breakpoints can also be set from the [Source] window.

The lines where you can set software PC breakpoints are marked by a bar "-" at the beginning of the lines. Move the cursor near the beginning of a line where you wish to set a breakpoint. The cursor will change shape to a circle (white ○). Click the mouse button there. The "-" mark at the beginning of the line will change to ■ (in red by default), indicating that a software PC breakpoint has been set. Clicking the button again clears the breakpoint you have set.

Otherwise, right-clicking at the "-" position will display a popup menu, so you can choose [Set Breakpoint] from the menu to set a software PC breakpoint. You can also choose [Set Temporary Breakpoint] from the menu and can set a temporary software PC breakpoint. When you have set a temporary software PC breakpoint, the color of the ■ mark that indicates the breakpoint becomes orange (by default).

For how to use the functions of the [Source] window, see Section 10.4.2, "[Source] Window".

- Notes:**
- Software PC breaks are implemented by an embedded `brk` instruction. Therefore, they cannot be used for the target board ROM in which instructions cannot be embedded. In such case, use hardware PC breaks instead.
  - When you set a software PC break at the address of an `ext`-based extended instruction or delayed branch instruction, note that you cannot set a breakpoint at other than the start address.
 

<code>ext xxxx</code>	... Can be set.	<code>jr*.d xxxx</code>	... Can be set.
<code>ext xxxx</code>	... Cannot be set.	Delayed instruction	... Cannot be set.
Extended instruction	... Cannot be set.		
  - The debugger refers to the memory map information set by loading a parameter file using the `c17 rpf` command as it checks each address to determine whether a software PC breakpoint can be set. Unless the `c17 rpf` command has been executed, the debugger does not perform error processing that pertains to the target system.

### Hardware PC breaks (`hbreak` and `thbreak` commands)

The on-chip debugger of the S1C17 Core is used to set the type of break. Breaks can also be simulated in simulator mode, as well as in other modes. When the executed PC address matches the address set by a command, the program is made to break before executing the instruction at that address. Hardware PC breakpoints can be set at one address location only.

Like software PC breaks, there are two types of hardware PC breaks: normal and temporary.

**`hbreak` command:** This command sets a normal hardware PC breakpoint.

Example: To set a hardware PC breakpoint at line 7 in `main.c`

```
(gdb) hbreak main.c:7
Hardware assisted breakpoint 1 at 0xc0001c: file main.c, line 7.
```

**`thbreak` command:** This command sets a temporary hardware PC breakpoint.

Example: To set a temporary hardware PC breakpoint at address `0xc0001e`

```
(gdb) thbreak *0xc0001e
Hardware assisted breakpoint 2 at 0xc0001e: file main.c, line 10.
```

When a hardware PC break occurs, the debugger waits for command input after displaying the following message:

```
(gdb) continue
Continuing.
```

```
Breakpoint 1, main () at main.c:7
```

Breakpoints can be set by specifying an address, source line number, or function/label name the same way as for software PC breakpoints.

Hardware PC breakpoints can also be set from the [Source] window.

The lines where you can set hardware PC breakpoints are marked by a bar "-" at the beginning of the lines. Right-clicking at the "-" position will display a popup menu, so you can choose [Set Hardware Breakpoint] from the menu to set a hardware PC breakpoint. The "-" mark at the beginning of the line will change to ■ (in blue by default), indicating that a hardware PC breakpoint has been set. Clicking the mouse button on the ■ clears the breakpoint you have set. Temporary hardware PC breakpoints cannot be set in the [Source] window.

For how to use the functions of the [Source] window, see Section 10.4.2, "[Source] Window".

### Breakpoint control (software PC break, hardware PC break)

When software PC breakpoints or hardware PC breakpoints are set, they are sequentially assigned break numbers beginning with 1 (regardless of the types of breaks set) that are displayed in a message in the [Console] window when you execute a break setup command. (See the examples above.) These numbers are required when you disable/enable or delete breakpoints individually at a later time. Even when you deleted breakpoints, the breakpoint numbers are not moved up (to reuse deleted numbers) until after you quit the debugger.

To manipulate the breakpoints you set, use the following commands:

**disable** command: This command disables a breakpoint. (Breakpoints are effective when set and remain effective unless disabled.)

Example: To disable breakpoint 1

```
(gdb) disable 1
```

**enable** command: This command enables a breakpoint.

Example: To enable breakpoint 1

```
(gdb) enable 1
```

**delete** or **clear** command: These commands delete a breakpoint.

Example 1: To delete breakpoints 1 and 2

```
(gdb) delete 1 2
```

Example 2: To delete a breakpoint at line 10 in main.c

```
(gdb) clear main.c:10
```

**ignore** command: This command specifies the number of times that a break is disabled.

Example: To specify that break 2 be disabled twice.

```
(gdb) ignore 2 2
```

Will ignore next 2 crossings of breakpoint 2.

**info breakpoints** command: This command displays a list of breakpoints.

Example: To display a list of breakpoints

```
(gdb) info breakpoints
```

```
Num Type          Disp Enb Address      What
1  breakpoint      keep y  0x00c00026  in main at main.c:11
   breakpoint already hit 1 time
   ignore next 10 hits
2  hw breakpoint   del  n  0x00c00038  in sub at main.c:20
```

For details, see the explanation of each command described later in this manual.

It is also possible to display a list of PC breakpoints in the [Breakpoints] window, where you can operate on breakpoints to disable/enable or delete. For details, see Section 10.4.6, "[Breakpoints] Window".

### Forcible break by the [Stop] button



[Stop] button

If the program has entered an endless loop or standby mode (HALT, SLEEP) and cannot exit that state, you can use the [Stop] button in the [Source] window to forcibly terminate the program. When this break occurs, the debugger waits for command input after displaying the following message:

```
Program received signal SIGTRAP, Trace/breakpoint trap.
```

By using the check box of the dialog box in which this message is displayed, you can choose not to display the message.

### Map breaks and breaks by invalid instruction execution (simulator mode)

The program is also made to break when accessing an invalid area.

- Notes:**
- The following breaks are only effective in simulator mode.
  - A memory map-related break occurs according to the memory map information set by a parameter file loaded by the `c17 rpf` command. An unexpected break may occur unless the loaded parameter file is correct.

#### Writes to the ROM area

The program is made to break after writing to the ROM area set by a parameter file. When this break occurs, the following message is output:

```
Break by writing ROM area.
```

#### Access to an undefined area

The program is made to break when accessing an undefined area other than those mapped by a parameter file.

```
Break by accessing no map.
```

#### Stack overflow

The program is made to break after writing to a stack exceeding the stack area set by a parameter file, thus causing it to overflow.

```
Break by stack overflow.
```

#### Execution of an invalid instruction

The program is made to break when executing an invalid instruction (not generated by the assembler).

```
Illegal instruction.
```

#### Execution of an invalid address instruction

The program is made to break when executing an instruction at an invalid address.

```
Illegal address exception.
```

#### Execution of an invalid delayed instruction

The program is made to break when executing an invalid delayed instruction.

```
Illegal delayed instruction.
```

## 10.6.7 Trace Functions

The debugger in simulator mode has a function to trace program execution.

**Note:** The trace function cannot be used in ICD Mini mode.

In simulator mode, you can use the `c17 tm` command to turn the trace function on or off, as well as specify the method of displaying data in a window or writing data to a file. When the trace function is turned on, trace results are displayed in a window or saved to a file for each instruction executed.

Example 1: To set the trace mode for displaying all information and to specify the `trace.log` file in which to save the information

```
(gdb) c17 tm on 0xff trace.log
```

Example 2: To turn trace mode off

```
(gdb) c17 tm off
```

num	clk	pc	code	bus addr/type/data	r0	r1	r2	r3	r4	r5	r6
1	6	004090	407e	-----	000000	000000	000000	000000	000000	000000	000000
2	7	004092	bc00	-----	000000	000000	000000	000000	000000	000000	000000
3	8	004094	4000	-----	000000	000000	000000	000000	000000	000000	000000
4	13	004096	18b9	003efc w32 00004098	000000	000000	000000	000000	000000	000000	000000
5	16	00420a	3e5c	003ef8 w32 00000000	000000	000000	000000	000000	000000	000000	000000
6	19	00420c	3edc	003ef4 w32 00000000	000000	000000	000000	000000	000000	000000	000000
7	20	00420e	9880	-----	000000	000000	000000	000000	000000	000000	000000
8	21	004210	4000	-----	000000	000000	000000	000000	000000	000000	000000
9	22	004212	4001	-----	000000	000000	000000	000000	000000	000000	000000
10	24	004214	d8b0	0000b0 w16 00000000	000000	000000	000000	000000	000000	000000	000000

The trace information displayed is listed below.

<Format of each trace information line>

*num clk pc code bus\_addr/type/data r0 r1 r2 r3 r4 r5 r6 r7 sp ie/il/cvzn src\_mix*

num: Number of executed instructions (in decimal)  
 Number of instructions executed since the CPU was reset

clk: Number of execution clocks (in decimal)  
 Number of execution clocks since the CPU was reset

pc: Address of executed instructions (in hexadecimal)

code: Instruction codes (in hexadecimal)

bus\_addr: Accessed memory addresses (in hexadecimal)

type: Type of bus operation  
 r8: Byte data read; r16: 16-bit data read; r32: 32-bit data read  
 w8: Byte data write; w16: 16-bit data write; w32: 32-bit data write

data: Read/written data (in hexadecimal)

r0-r7: r0-r7 register values (in hexadecimal)

sp: sp register value (in hexadecimal)

ie: IE bit value in psr

il: IL bit value in psr

cvzn: C, V, Z and N bit values in psr

src\_mix: Disassembled contents and source codes of executed instructions

The trace information is displayed in the [Trace] window when you choose to display in a window by using the `c17 tm` command. When you choose to save to a file, the information is output to a file, and not displayed in a window.

**Note:** The number of clock cycles executed (clk) displayed in a window is calculated using the wait cycle information set in a parameter file. The displayed information may not be correct if the parameter file was erroneously set.

## 10.6.8 Simulated I/O

The simulated I/O function of **gdb** allows you to evaluate the external input/output of the serial interface, etc. by means of standard input/output (stdin, stdout) or file input/output.

### Input by stdin (c17 stdin command)

Set the following conditions in the **c17 stdin** command:

- Break address
- Input buffer address (with buffer size fixed to 65 bytes)
- Input file (when omitted, input from [Simulated I/O] window)

After setting these conditions, run the program continuously.

#### When an input file is specified

When the set break address is reached, **gdb** reads data from the specified file and places it in the buffer. Then it resumes executing the program from the address where it left off.

Example 1: To set a data input function

```
(gdb) c17 stdin 1 READ_FLASH_READ_BUF input.txt
```

Example 2: To turn the data input function off

```
(gdb) c17 stdin 2
```

#### When no input files are specified

When the set break address is reached, **gdb** opens the [Simulated I/O] window and waits for data to be entered from the keyboard. When you enter data and press the [Enter] key, **gdb** writes the data entered to the buffer and resumes executing the program from the address where it left off.

Example: To set a data input function using the [Simulated I/O] window

```
(gdb) c17 stdin 1 READ_FLASH_READ_BUF
```

### Output by stdout (c17 stdout command)

Set the following conditions in the **c17 stdout** command:

- Break address
- Output buffer address (with buffer size fixed to 65 bytes)
- Output file (when omitted, output to [Simulated I/O] window)

After setting these conditions, run the program continuously.

#### When an output file is specified

When the set break address is reached, **gdb** outputs the contents of the buffer to the specified file. Then it resumes executing the program from the address where it left off.

Example 1: To set a data output function

```
(gdb) c17 stdout 1 WRITE_FLASH_WRITE_BUF output.txt
```

Example 2: To turn the data output function off

```
(gdb) c17 stdout 2
```

#### When no output files are specified

When the set break address is reached, **gdb** opens the [Simulated I/O] window and displays the buffer contents in that window. Then it resumes executing the program from the address where it left off.

Example: To set a data output function using the [Simulated I/O] window

```
(gdb) c17 stdout 1 WRITE_FLASH_WRITE_BUF
```

## Requirements for the program

Before the simulated I/O function can be used, the following must be defined in the program:

### Definition of input/output buffers

Before using the program, define global buffers that **gdb** will use for data input/output in the format shown below.

Definition of an input buffer: `unsigned char READ_BUF[65]`

Definition of an output buffer: `unsigned char WRITE_BUF[65]`

For the buffer name, use any name conforming to symbol name conventions. Fix the buffer size to 65 bytes. Use this symbol name to specify the buffer address when executing the `c17 stdin` and `c17 stdout` commands.

When data is entered, the size of the actually entered data (1 to 64 bytes) is placed in `READ_BUF[0]`. If EOF is entered, value 0 is placed in `READ_BUF[0]`. The input data is stored in `READ_BUF[1]` and those that follow. To output data, write the size of data to be output (1–64) to `WRITE_BUF[0]`, then output the data to `WRITE_BUF[1]` and those that follow. To output EOF, write value 0 to `WRITE_BUF[0]`. A data row of up to 64 bytes in size can be input/output between **gdb** and the program.

### Definition of data-updating global labels

Before using the program, define global labels like those shown below at the position where **gdb** inputs data to the input buffer and at the position where **gdb** outputs data from the output buffer.

Input position: `.global READ_FLASH`  
`READ_FLASH:`

Output position: `.global WRITE_FLASH`  
`WRITE_FLASH:`

Any name can be used for the labels. Use this symbol name to specify the break address when executing the `c17 stdin` and `c17 stdout` commands.

In the C source, define these labels in the lower-level `write` and `read` functions among the standard input/output library functions (see Section 7.3.4).

For examples of actual programs, refer to the sample programs and debugger command file installed in the `\gnu17\sample\S1C17common\simulator\simulatedIO` directory.

When the program breaks at `READ_FLASH`, **gdb** reads data from a file and loads the data into the defined input buffer. Then it resumes executing the program. When the program breaks at `WRITE_FLASH`, **gdb** outputs data from the output buffer to a file, then resumes executing the program.

## Precautions

- The break addresses specified in the `c17 stdin` and `c17 stdout` commands cannot duplicate those of software PC breaks.
- Because the debugger uses software PC breaks internally, addresses in the target board ROM cannot be specified.
- Only ASCII characters can be used for input/output. Binary data (especially 0x0 and 0x1a) may cause the CPU to operate erratically.
- The parts of the program where `c17 stdin` or `c17 stdout` perform input/output must be executed successively by the `continue` command (do not execute in single stepping). Also make sure that no breaks will occur in areas near those parts.

## 10.6.9 Flash Memory Operation

The debugger **gdb** has a function to manipulate flash memory mounted in the target board, as well as the flash write function to use the ICD.

### Manipulating flash memory on the target board

The debugger **gdb** has a utility and commands that allow you to write data to or erase data from flash memory built into the S1C17 chip or mounted on the target board. This utility and these commands can be used in the debugging environments of ICD Mini mode.

Follow the procedure described below to write data to flash memory. For more details, refer to `readme.txt` for flash support utility **fls17**.

#### 1. Loading flash routines

Use the `load` command to load flash routines (erase and write routines) into internal RAM, etc.

Example:

```
(gdb) file 291v800t.elf           (Load debugging information.)
(gdb) target icd usb             (Connect the target.)
C17 ICD17 debugging
Connecting with target (ID_OK) .... done
ICD Initializing (ID_INITIALIZE) ... done
Read ICD Version (ID_VER_READ) .... done
  ICD hardware version ..... 4.1
  ICD software version ..... 10.0
Debug base address (ID_DATA_READ) .. 0x003F00
Boot address (ID_DATA_READ) ..... 0x004090
Target file is pointer24.
(gdb) load                       (Load flash routines.)
```

Use these routines when actually erasing and writing data.

#### 2. Setting flash memory (`c17 fls` command)

Set the start and end addresses of flash memory, along with the entry addresses of erase and write routines loaded in step 1 in **gdb**.

Example: When the flash memory area is `0xc00000` to `0xcfffff` and the entry addresses of the erase and write routines are `FLASH_ERASE` and `FLASH_LOAD`

```
(gdb) c17 fls 0xc00000 0xcfffff FLASH_ERASE FLASH_LOAD
```

#### 3. Erasing flash memory (`c17 fle` command)

Erase the entire area or sectors of flash memory. The contents of flash memory will be changed to `0xff`.

Example: To erase the entire area of flash memory whose address is `0xc00000` in the flash memory control register

```
(gdb) c17 fle 0xc00000 0 0
```

Always be sure to execute the `c17 fle` command after the `c17 fls` command. When you do not wish to erase flash memory, specify the sector range ("0 0" in the examples above) as "-1 0". Only the control register will be modified.



#### 4. Writing to flash memory

Use the `load` command to write a program to flash memory.

Example:

```
(gdb) load sample.elf
```

Use the `set` command to write data to flash memory.

Example: To write 0x1234 to address 0xc00002

```
(gdb) set {short}0xc00002 = 0x1234
```

```
(gdb) x /8h 0xc00000
```

```
C00000: FFFF 1234 FFFF FFFF FFFF FFFF FFFF FFFF
```

For 16-bit devices, use the `set {short}` command. Rewriting in units of bytes is not supported. The written contents can be confirmed using the `x` command.

The data to be entered in flash memory set by the `c17 fls` command in step 2 is passed to the flash write routine, by which the data is written to flash memory. This is an exception and all other operations are processed as writing to RAM. If flash memory has not been erased (not 0xff), an error is returned.

#### Flash writer function of the ICD Mini (S5U1C17001H)

The ICD Mini (S5U1C17001H) incorporates a flash writer function, and **gdb** has commands to control this function.

For how to use the ICD Mini (S5U1C17001H) as a flash writer, refer to the "S5U1C17001H Manual (S1C17 Family In-Circuit Debugger)".

The flash writer control commands can only be used with the ICD Mini (S5U1C17001H) in ICD Mini mode, as described below.

**Note:** The ICD board does not support the flash writer function.

#### Erasing programs/data (`c17 fwe` command)

The `c17 fwe` command erases the data erase/write program or write data and address information loaded in the S5U1C17001H.

Example 1: To erase write data

```
(gdb) c17 fwe 0
```

Example 2: To erase the data erase/write program

```
(gdb) c17 fwe 1
```

#### Loading a program (`c17 fwlp` command)

The `c17 fwlp` command loads the data erase/write program from the host in the ICD Mini (S5U1C17001H) and sets entry information about the erase/write routines.

Example: When the data erase/write program file is `writer.sa` and the start addresses of erase and write routines are 0x90 and 0xb4, respectively

```
(gdb) c17 fwlp writer.sa 0x90 0xb4
```

#### Loading data (`c17 fwld`, `c17 fwdc` commands)

The `c17 fwld` command loads the data to be written to flash memory from the host in the ICD Mini (S5U1C17001H). The `c17 fwdc` command loads the data saved in target board memory into the ICD Mini (S5U1C17001H). Also set the range of flash memory to be erased.

Example 1: To load `sample.sa` after specifying that all blocks are to be erased

```
(gdb) c17 fwld sample.sa 0 0 0
```

Example 2: To load 1MB of data from target memory address `FLASH_START` after specifying that all blocks are to be erased

```
(gdb) c17 fwdc FLASH_START 0x100000 0 0 0
```

**Displaying flash writer information (c17 fwd command)**

The **c17 fwd** command displays information about the data loaded in the ICD Mini (S5U1C17001H) and information about the erase/write program.

```
(gdb) c17 fwd
CPU data address      : xxxxxxxx
Data size            : xxxxxxxx
Erase start block    : xxxxxxxx
Erase end block      : xxxxxxxx
Erase parameter      : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

CPU program address  : xxxxxxxx
Program size         : xxxxxxxx
Erase routine entry  : xxxxxxxx
Write routine entry  : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

## 10.6.10 Support for Big Endian

The tools (C compiler to the linker) and libraries only support little endian. Note that the C compiler cannot create elf files that can be loaded in big endian areas. Regarding data references, however, the debugger supports big endian.

### Method of specifying big endian (in simulator mode)

To set information about big endian areas in the debugger while in simulator mode, specify it in the map information of a parameter file. For areas you wish to set to big endian, write B in the 5th parameter. Without this statement, areas are assumed to be little endian. However, the S1C17 chip for which you are developing software must be a type that supports big endian. Also note that internal ROM, RAM, and I/O cannot be set to big endian.

For details about parameter files, see Section 10.8, "Parameter Files".

### Operation of debugger commands

#### Memory manipulating commands (`x, set, c17 fg/fh/fw, c17 mvb/mvh/mvw`)

These commands perform operations and display information suitable for endian format because memory is accessed for read and write in units of bytes, 16 bits, or 32 bits according to the type of data in each command.

#### Load command

Swaps data suitable for endian format and writes data in units of 16 bits. For this reason, programs created by the C compiler cannot be loaded correctly in big endian areas.

## 10.7 Command Reference

### 10.7.1 List of Commands

Table 10.7.1.1 List of commands

Classification	Command	Operation	Supported modes		Page
			ICD Mini	SIM	
Memory manipulation	<b>c17 fb</b>	Fill area (in bytes)	○	○	10-62
	<b>c17 fh</b>	Fill area (in 16 bits)	○	○	10-62
	<b>c17 fw</b>	Fill area (in 32 bits)	○	○	10-62
	<b>x /b</b>	Memory dump (in bytes)	○	○	10-64
	<b>x /h</b>	Memory dump (in 16 bits)	○	○	10-64
	<b>x /w</b>	Memory dump (in 32 bits)	○	○	10-64
	<b>set {char}</b>	Data input (in bytes)	○	○	10-66
	<b>set {short}</b>	Data input (in 16 bits)	○	○	10-66
	<b>set {long}</b>	Data input (in 32 bits)	○	○	10-66
	<b>c17 mvb</b>	Copy area (in bytes)	○	○	10-67
	<b>c17 mvh</b>	Copy area (in 16 bits)	○	○	10-67
	<b>c17 mvw</b>	Copy area (in 32 bits)	○	○	10-67
	<b>c17 df</b>	Save memory contents	○	○	10-69
	<b>c17 readmd</b>	Memory read mode	○	–	10-71
Register manipulation	<b>info reg</b>	Display register	○	○	10-72
	<b>set \$</b>	Modify register	○	○	10-73
Program execution	<b>continue</b>	Execute continuously	○	○	10-74
	<b>until</b>	Execute continuously with temporary break	○	○	10-76
	<b>step</b>	Single-step (every line)	○	○	10-78
	<b>stepi</b>	Single-step (every mnemonic)	○	○	10-78
	<b>next</b>	Single-step with skip (every line)	○	○	10-80
	<b>nexti</b>	Single-step with skip (every mnemonic)	○	○	10-80
	<b>finish</b>	Quit function	○	○	10-82
	<b>c17 callmd</b>	Set user function call mode	○	○	10-83
	<b>c17 call</b>	Call user function	○	○	10-84
CPU reset	<b>c17 rst</b>	Reset (execute reset.gdb)	○	○	10-86
	<b>c17 rstt</b>	Reset target	○	–	10-87
Interrupt	<b>c17 int</b>	Interrupt	–	○	10-88
	<b>c17 intclear</b>	Clear interrupt	–	○	10-89
	<b>c17 int_load</b>	Load interrupt event file	–	○	10-90
Break	<b>break</b>	Set software PC break	○	○	10-91
	<b>tbreak</b>	Set temporary software PC break	○	○	10-91
	<b>hbreak</b>	Set hardware PC break	○	○	10-94
	<b>thbreak</b>	Set temporary hardware PC break	○	○	10-94
	<b>delete</b>	Clear break by break number	○	○	10-97
	<b>clear</b>	Clear break by break position	○	○	10-99
	<b>enable</b>	Enable breakpoint	○	○	10-101
	<b>disable</b>	Disable breakpoint	○	○	10-101
	<b>ignore</b>	Disable breakpoint with ignore counts	○	○	10-103
	<b>info breakpoints</b>	Display breakpoint list	○	○	10-104
	<b>c17 timebrk</b>	Set lapse of time break	○	–	10-105
Symbol information	<b>info locals</b>	Display local symbol	○	○	10-106
	<b>info var</b>	Display global symbol	○	○	10-106
	<b>print</b>	Alter symbol value	○	○	10-107
File loading	<b>file</b>	Load debugging information	○	○	10-108
	<b>load</b>	Load program	○	○	10-109
	<b>c17 loadmd</b>	Set program load mode	○	–	10-110
Map information	<b>c17 rpf</b>	Set map information	–	○	10-111
	<b>c17 map</b>	Display map information	–	○	10-112
Flash memory manipulation	<b>c17 fls</b>	Set flash memory	○	–	10-113
	<b>c17 fle</b>	Erase flash memory	○	–	10-114
Trace	<b>c17 tm</b>	Set trace mode	–	○	10-115
Simulated I/O	<b>c17 stdin</b>	Data input simulation	○	○	10-118
	<b>c17 stdout</b>	Data output simulation	○	○	10-119

Classification	Command	Operation	Supported modes		Page
			ICD Mini	SIM	
Flash writer	<b>c17 fwe</b>	Erase program/data	● *1	–	10-120
	<b>c17 fwlp</b>	Load program	● *1	–	10-121
	<b>c17 fwld</b>	Load data	● *1	–	10-122
	<b>c17 fwdc</b>	Copy target memory	● *1	–	10-123
	<b>c17 fwd</b>	Display flash writer information	● *1	–	10-124
Other	<b>c17 log</b>	Logging	○	○	10-125
	<b>source</b>	Execute command file	○	○	10-126
	<b>c17 clockmd</b>	Set execution counter mode	○	○	10-127
	<b>c17 clock</b>	Display execution counter	○	○	10-127
	<b>target</b>	Connect target	○	○	10-129
	<b>detach</b>	Disconnect target	○	○	10-130
	<b>pwd</b>	Display current directory	○	○	10-131
	<b>cd</b>	Change current directory	○	○	10-131
	<b>c17 firmupdate</b>	Update ICD firmware	○	–	10-132
	<b>c17 ttbr</b>	Set TTBR	–	○	10-133
	<b>c17 help</b>	Help	○	○	10-134
	<b>quit</b>	Quit debugger	○	○	10-136

Supported modes: ○ = Can be used, – = Cannot be used, ● = Conditional

\*1: These commands can be used with the ICD Mini (S5U1C17001H). They cannot be executed normally with the ICD board (an error message will not be displayed).

## 10.7.2 Detailed Description of Commands

This chapter describes in detail each debugger command using the format shown below.

### **Command name** (operation of command) [Supported modes]

A detailed description of each command begins with the command name in this format.

[Supported modes] shows such modes as ICD Mini and SIM in which the command can be used. You cannot use the command in modes other than those written here.

Basically, each command is described separately. However, two or more commands (belonging to the same operation group) that differ only slightly or which can be better understood when explained together are described collectively.

#### Operation

Explains the operation of the command.

#### Format

Shows the format in which the command is entered in the [Console] window and the contents of parameters. Parameters enclosed in brackets [ ] can be omitted. Otherwise, no parameters can be omitted. The *italicized* characters denote parameters specified with numeric values or symbols.

#### Usage example

Shows an example of how to enter the command and the results of command execution, etc.

#### GUI

Shows the method (if any) of executing the command other than entering it in the [Console] window. When this item is not indicated, the command can only be executed in the [Console] window or a command file.

#### Notes

Describes limitations on use of the command or precautions to be taken when using the command.

Some commands have additional items other than those described above when needed for explanatory purposes.

## 10.7.3 Memory Manipulation Commands

**c17 fb** (fill area, in bytes)

**c17 fh** (fill area, in 16 bits)

**c17 fw** (fill area, in 32 bits)

[ICD Mini / SIM]

### Operation

**c17 fb** Rewrites specified memory area with specified byte data.

**c17 fh** Rewrites specified memory area with specified 16-bit data.

**c17 fw** Rewrites specified memory area with specified 32-bit data.

### Format

**c17 fb** *StartAddr EndAddr Data*

**c17 fh** *StartAddr EndAddr Data*

**c17 fw** *StartAddr EndAddr Data*

*StartAddr*: Start address of area to be filled (decimal, hexadecimal, or symbol)

*EndAddr*: End address of the area to be filled (decimal, hexadecimal, or symbol)

*Data*: The data to write (decimal or hexadecimal)

Conditions:  $0 \leq \text{StartAddr} \leq \text{EndAddr} \leq 0\text{xffffffff}$ ,  $0 \leq \text{Data} \leq 0\text{xff}$  (c17 fh),  $0 \leq \text{Data} \leq 0\text{xffff}$  (c17 fh),  
 $0 \leq \text{Data} \leq 0\text{xffffffff}$  (c17 fw)

### Usage example

#### ■ Example 1

```
(gdb) c17 fb 0x0 0xf 0x1
```

Start address = 0x0, End address = 0xf, Fill data = 0x1 .....done

```
(gdb) x /16b 0x0 (memory dump command)
```

```
0x0: 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
```

```
0x8: 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
```

The entire memory area from address 0x0 to address 0xf is rewritten with byte data 0x01.

#### ■ Example 2

```
(gdb) c17 fh 0x0 0xf 0x1
```

Start address = 0x0, End address = 0xe, Fill data = 0x1 .....done

```
(gdb) x /8h 0x0 (memory dump command)
```

```
0x0: 0x0001 0x0001 0x0001 0x0001 0x0001 0x0001 0x0001 0x0001
```

The entire memory area from address 0x0 to address 0xf is rewritten with 16-bit data 0x0001. (This applies to when using little endian.)

#### ■ Example 3

```
(gdb) c17 fw 0x0 0xf 0x1
```

Start address = 0x0, End address = 0xc, Fill data = 0x1 .....done

```
(gdb) x /4w 0x0 (memory dump command)
```

```
0x0: 0x00000001 0x00000001 0x00000001 0x00000001
```

The entire memory area from address 0x0 to address 0xf is rewritten with 32-bit data 0x00000001. (This applies to when using little endian.)

## Notes

- Writing in units of 16 bits or 32 bits is performed in little endian format. However, when debugging a program in simulator mode, you can set a specified area to big endian format in a parameter file.
- Even if the entire or a portion of the memory section specified for write is an unused area, no errors are assumed. Data is rewritten, except in unused areas.
- The data write memory section is aligned to boundary addresses according to the size of data.  
(gdb) c17 fw 0x3 0x9 0x0

For example, when a write memory section is specified as shown above, and because start address 0x3 and end address 0x9 are not located on 32-bit data boundaries, both are aligned to boundary addresses by setting the 2 low-order bits to 00 (LSB = 0 for 16 bits). The following shows the actually executed command, where 32-bit data addresses 0x0 to 0x8 (byte data addresses 0x0 to 0xb) are rewritten with data 0x00000000.

```
(gdb) c17 fw 0x0 0x8 0x0
```

- If the specified address exceeds the 24-bit range, an error is assumed.
- Data parameters are only effective for the 8 low-order bits for c17 fb, 16 low-order bits for c17 fh, and 32 low-order bits for c17 fw, with excessive bits being ignored. For example, when data 0x100 is specified in c17 fb, it is processed as 0x00.
- If the end address is smaller than the start address, an error is assumed.
- Even when memory contents are modified by this command, the contents displayed in the [Memory] and [Source] windows are not updated. Therefore, perform the appropriate operation to update display in each window. Similarly, even when the program area is rewritten, the source displayed in a window remains unchanged.

**X** (memory dump)

[ICD Mini / SIM]

**Operation**

Dumps memory contents (in hexadecimal) to a window. The data size, display start address, and display data counts can be specified.

**Format**

**x** [ / [*Length*] *Size*] [*Address*]

*Length*: Number of data items to display (in decimal)  
1 when omitted.

*Size*: One of the following symbols that specify data size (in which units of data are displayed)  
**b** In units of bytes  
**h** In units of 16 bits  
**w** In units of 32 bits (default)

*Address*: Address from which to start displaying data (decimal, hexadecimal, or symbol)  
When omitted, the last address displayed when previously executing the **x** command is assumed.  
The default address assumed at **gdb** startup is 0x0.

Conditions:  $0 \leq \textit{Length} \leq 0\text{xffffffff}$ ,  $0 \leq \textit{Address} \leq 0\text{xffff}$

**Display**

Memory contents are displayed as described below.

*Address*[<*Symbol*>]: *Data* [*Data* ...]

*Address*: The start address of each line of data is displayed in hexadecimal.

*Symbol*: When the address displayed at the beginning of a line has a symbol or label defined for it, the name of that symbol or label is displayed. When an intermediate address of a function or variable is specified, the specified symbol and a decimal offset (<*Symbol* + *n*>) are also displayed.

*Data*: Up to 16 bytes of data starting from *Address* are displayed on one line.

**Usage example**

## ■ Example 1

```
(gdb) x
0x0: 0x00000000
```

When all parameters are omitted after startup, the command is executed as "**x /1w 0x0**".

## ■ Example 2

```
(gdb) x /b 0
0x0 <i>: 0xe3
(gdb) x /b 1
0x1 <i+1>: 0xa1
```

When *Size* is specified but *Length* omitted, one unit of data equal to the specified data size is displayed. The letter *i* is a symbol defined at address 0x0. If any address other than the address at the beginning of a variable, etc. is specified, <*symbol+offset*> is displayed as the symbol.

## ■ Example 3

```
(gdb) x /16h _START_text
0xc00000 <_START_text>: 0x0004 0x00c0 0xc020 0x6c0f 0xa0f1 0xc000 0xc000 0x6c0f
0xc00010 <boot+12>: 0xc000 0xc000 0x1c04 0xdff8 0xdfff 0x1ef5 0x0200 0x6c04
```

When *Length* is specified, the specified amount of data is displayed. When a code area is displayed, <*label+offset*> is displayed as the symbol, even for addresses with no symbols defined as in ASSEMBLY display of the [Source] window.



#### ■ Example 4

```
(gdb) x /4w 0
0x0 <i>: 0x00001ae3 0x00000000 0x00000000 0x00000000
(gdb) x
0x10: 0x00000000
(gdb) x
0x14: 0x00000000
```

When the `x` command is executed once, you can dump and display a single unit of data (having the same size as that of the previous address) from the address following the previous address by simply entering `x`.

#### ■ Example 5

```
(gdb) x /w &i
0x0 <i>: 0x00000010
(gdb) x /w i
0x10: 0x00000000
```

When specifying an address with a data symbol that references the assigned address, add `&` when you enter the command. When only specifying a symbol, note that its data value is used as the address. In such case, `&` need not be added because labels in program code indicate assigned addresses.

#### GUI

The contents of memory can be confirmed in the [Memory] window. (See 10.4.5, "[Memory] Window".)

#### Notes

- Memory contents are displayed in little endian format. However, when debugging a program in simulator mode, you can set a specified area to big endian format in a parameter file.
- Even when an unused area of memory is specified, no errors are assumed. However, the displayed data is not valid.
- Even if the specified address is not a boundary address conforming to the data size, the `x` command starts displaying memory contents from that address.
- If the specified address exceeds the 24-bit range, an error is assumed.
- Executing this command does not affect the [Memory] window.

**set { }** (data input)

[ICD Mini / SIM]

**Operation**

Writes specified data to a specified address.

**Format**

**set {Size}Address=Data**

*Size:* One of the following symbols that specify data size

**char** In units of bytes  
**short** In units of 16 bits (default)  
**int** In units of 16 bits  
**long** In units of 32 bits

*Address:* Address to which to write data (decimal, hexadecimal, or symbol)

*Data:* The data to write (decimal, hexadecimal, or symbol)

*Conditions:*  $0 \leq \text{Address} \leq 0\text{xfffff}$ ,  $0 \leq \text{Data} \leq 0\text{xff}$  (set {char}),  $0 \leq \text{Data} \leq 0\text{xffff}$  (set {short/int}),  
 $0 \leq \text{Data} \leq 0\text{xffffffff}$  (set {long})

**Usage example**

## ■ Example 1

```
(gdb) set {char}0x1000=0x55
(gdb) x /b 0x1000
0x1000: 0x55
```

Byte data 0x55 is written to address 0x1000.

## ■ Example 2

```
(gdb) set {short}0x1000=0x5555
(gdb) x /h 0x1000
0x1000: 0x5555
```

16-bit data 0x5555 is written to address 0x1000.

## ■ Example 3

```
(gdb) set {long}&i=0x55555555
(gdb) x /w &i
0x0 <i>: 0x55555555
```

32-bit data 0x55555555 is written to long variable i.

**GUI**

Memory contents can be changed in the [Memory] window. (See 10.4.5, "[Memory] Window".)

**Notes**

- Writing in units of 16 bits or 32 bits is performed in little endian format. However, when debugging a program in simulator mode, you can set a specified area to big endian format in a parameter file.
- Even when an unused area of memory is specified, no errors are assumed.
- If the specified address exceeds the 24-bit range, an error is assumed.
- Data parameters are only effective for the 8 low-order bits for set {char}, 16 low-order bits for set {short} and set {int}, and 32 low-order bits for set {long}, with excessive bits being ignored. For example, when data 0x100 is specified in set {char}, it is processed as 0x00.
- Even when memory contents are modified by this command, the contents displayed in the [Memory] and [Source] windows are not updated. Therefore, perform the appropriate operation to update display in each window. Similarly, even when the program area is rewritten, the source displayed in a window remains unchanged.

**c17 mvb** (copy area, in bytes)

**c17 mvh** (copy area, in 16 bits)

**c17 mvw** (copy area, in 32 bits)

[ICD Mini / SIM]

#### Operation

- c17 mvb** Copies the content of a specified memory area to another area in units of bytes.
- c17 mvh** Copies the content of a specified memory area to another area in units of 16 bits.
- c17 mvw** Copies the content of a specified memory area to another area in units of 31 bits.

#### Format

**c17 mvb** *SourceStart SourceEnd Destination*  
**c17 mvh** *SourceStart SourceEnd Destination*  
**c17 mvw** *SourceStart SourceEnd Destination*

*SourceStart*: Start address of area from which to copy (decimal, hexadecimal, or symbol)

*SourceEnd*: End address of area from which to copy (decimal, hexadecimal, or symbol)

*Destination*: Start address of area to which to copy (decimal, hexadecimal, or symbol)

Conditions:  $0 \leq \textit{SourceStart} \leq \textit{SourceEnd} \leq 0\text{xfffff}$ ,  $0 \leq \textit{Destination} \leq 0\text{xfffff}$

#### Usage example

##### Example 1

```
(gdb) x /16b 0
0x0: 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
0x8: 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f
(gdb) c17 mvb 0x0 0x7 0x8
Start address = 0x0, End address = 0x7, Destination address = 0x8 .....done
(gdb) x /16b 0
0x0: 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
0x8: 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
```

The content of a memory area specified by addresses 0x0 to 0x7 is copied to an area beginning with address 0x8.

##### Example 2

```
(gdb) x /4w 0
0x0 <i>: 0x00000000 0x11111111 0x22222222 0x33333333
(gdb) c17 mvw i i i+4
Start address = 0x0, End address = 0x0, Destination address = 0x4 .....done
(gdb) x /4w 0
0x0 <i>: 0x00000000 0x00000000 0x22222222 0x33333333
```

The content of long variable *i* is copied to an area located four bytes after that int variable.

#### Notes

- When the source and destination have different endian formats, the data formats are converted when copied from the source to the destination.
- If the specified address exceeds the 24-bit range, an error is assumed.
- In **c17 mvh** and **c17 mvw**, addresses are adjusted to boundary addresses conforming to the data size. This is accomplished by processing the LSB address bit as 0 for **c17 mvh** and the 2 low-order address bits as 00 for **c17 mvw**.
- If the end address at the source is smaller than its start address, an error is assumed.
- If a specified memory section at the source contains an unused area, the data in that area is handled as 0xf0 when copied.
- If a memory section at the destination contains an unused area, data is only copied to the effective area (excluding the unused area).

## 10 DEBUGGER

- When the start address at the destination is smaller than that of the source, data is copied sequentially beginning with the start address. Conversely, when the start address at the destination is larger than that of the source, data is copied sequentially beginning with the end address. Therefore, data is always copied even when the specified destination address exists within the source area.
- If the end address at the destination exceeds 0xfffff, data is only copied only up to 0xfffff.
- Even when memory contents are modified by this command, the contents displayed in the [Memory] and [Source] windows are not updated. Therefore, perform the appropriate operation to update display in each window. Similarly, even when the program area is rewritten, the source displayed in a window remains unchanged.



### ■ Example 4

```
(gdb) c17 df 0x1000 0x1fff 3 dump.mot a ; Footer is not output. (First)
(gdb) c17 df 0x3000 0x3fff 3 dump.mot a ; Footer is not output.
(gdb) c17 df 0x5000 0x5fff 3 dump.mot a ; Footer is not output.
(gdb) c17 df 0x7000 0x7fff 3 dump.mot f ; Footer is output. (Last)
```

The contents of addresses 0x1000–0x7fff (every 0x1000 addresses) are written out in Motorola S3 format to the file "dump.mot".

If no *Append* parameters exist or the parameter 'f' is specified, a footer record is output to a Motorola S3 format file.

#### Notes

- If the specified address exceeds the 24-bit range, an error is assumed.
- If the end address is smaller than the start address, an error is assumed.

**c17 readmd** (memory read mode)

[ICD Mini]

**Operation**

Selects the method for reading data from the target memory from the following two modes:

## 1. Normal mode

Memory data is always read in bytes.

## 2. Boundary exact mode

Data is read using the appropriate instruction in the mini-monitor's external routine from the correct boundary address aligned with access size. The `ld.b` instruction is used to access memory in 8-bit units.

The `ld` instruction is used to access memory in 16 bit or 32-bit units.

Use boundary exact mode if reading data in units of bytes every time becomes inconvenient.

**Format****c17 readmd Mode**

*Mode:*       Selects memory read mode  
           0   Normal mode (default)  
           1   Boundary exact mode

**Usage example**

```
(gdb) c17 readmd 1
```

Selects boundary exact mode.

**Notes**

- This command cannot be used in simulator mode.
- Selecting boundary exact mode will slow memory reading.
- Memory read mode settings affect the reading of data from memory by the commands listed below.  
`c17 df, x`

## 10.7.4 Register Manipulation Commands

### info reg (display register)

[ICD Mini / SIM]

#### Operation

Displays the contents of the CPU registers.

#### Format

**info reg** [*RegisterName*]

*RegisterName*: Name of register to display (specified in lowercase letters)

r0-r7, sp, pc, psr

If the above is omitted, the contents of all registers are displayed.

#### Display

Register contents are displayed as described below.

*Register*    *Hexadecimal*    *Decimal*

*Register*:        This is a register name.

*Hexadecimal*: Shows the register value in hexadecimal.

*Decimal*:        Shows the register value in decimal.

#### Usage example

##### ■ Example 1

```
(gdb) info reg r1
r1                0xaaaaaa    1184810
(gdb) info reg pc
pc                0x4090     16528
```

When a register name is specified, only the content of that register is displayed.

##### ■ Example 2

```
(gdb) info reg
r0                0xd20      3360
r1                0xaaaaaa    1184810
r2                0xaaaaaa    1184810
r3                0xaaaaaa    1184810
r4                0x690      1680
r5                0xaaaaaa    1184810
r6                0x0        0
r7                0xaaaaaa    1184810
sp                0x7f8      2040
pc                0xc00030   12582960
psr               0x7f8      2040
```

#### GUI

The contents of all registers can be confirmed in the [Registers] window. (See 10.4.4, "[Registers] Window".)

#### Notes

Be sure to specify register names in lowercase letters. Using uppercase letters for register names or specifying nonexistent register names results in an error.



**set \$** (modify register)

[ICD Mini / SIM]

**Operation**

Changes the values of the CPU registers.

**Format**

**set \$RegisterName=Value**

*RegisterName*: Name of register to change (specified in lowercase letters)

r0-r7, sp, pc, psr

*Value*: 24-bit data to set in the register (decimal, hexadecimal, or symbol)

Conditions:  $0 \leq \textit{Value} \leq 0\text{xfffff}$

**Usage example**

```
(gdb) set $r1=0x10000
(gdb) info reg r1
r1          0x10000      65536
(gdb) set $pc=main
```

In addition to numerals, symbols can also be used to set values.

**GUI**

Register contents can also be modified in the [Registers] window. (See 10.4.4, "[Registers] Window".)

**Notes**

- If the specified value exceeds the 24-bit range, the 24 low-order bits only will be effective.
- The contents of the set values are not checked internally. No errors are assumed even when values other than 16-bit or 32-bit boundary addresses are specified for PC or SP, respectively. However, when the registers are actually modified, values are forcibly adjusted to boundary addresses by truncating the lower bits.
- The contents displayed in the [Registers] window are not updated by executing this command.

## 10.7.5 Program Execution Commands

### **continue** (execute continuously)

[ICD Mini / SIM]

#### Operation

Executes the target program from the current PC address.

The program is run continuously until it is made to break by one of the following causes:

- Already set break conditions are met.
- The [Stop] button is clicked.

When reexecuting a target program halted because break conditions have been met, you can specify to disable the current breakpoint the specified number of times.

#### Format

**continue** [*IgnoreCount*]  
**cont** [*IgnoreCount*] (abbreviated form)

*IgnoreCount*: Specifies the number of breaks (decimal or hexadecimal)

The program is run continuously until break conditions are met the specified number of times.

#### Usage example

##### ■ Example 1

```
(gdb) continue
Continuing.
```

```
Breakpoint 1, main () at main.c:13
```

When `continue` is executed with *IgnoreCount* omitted, the target program starts running from the current PC address and stops the first time break conditions are met.

##### ■ Example 2

```
(gdb) cont 5
Will ignore next 4 crossings of breakpoint 1. Continuing.
```

```
Breakpoint 1, main () at main.c:13
```

Because value 5 is specified for *IgnoreCount*, break conditions that have been met four times (= 5 - 1) since the program started running are ignored, and the program breaks when break conditions are met the fifth time. In this example, the target program is restarted after being halted at the PC breakpoint (break 1) set at line 13 in `main.c`, and the program stops upon the fifth hit at that PC breakpoint.

The same effect is obtained by executing the following command:

```
(gdb) ignore 1 4
Will ignore next 4 crossings of breakpoint 1.
(gdb) continue
```

#### GUI

In addition to the above, the `continue` command can be executed using one of the following methods. In such case, however, you cannot specify *IgnoreCount*.

- Click the [Continue] button in the [Source] window.



[Continue] button

- Choose [Continue] from the [Control] menu in the [Source] window.
- Enter [c] ([C]) while the [Source] window is open.

When the target program is executed by the `continue` command, the contents displayed in the [Source] and [Registers] windows are updated in real time. The [Memory] window is not updated.

## Notes

- To run the program from the beginning, execute `c17 rst` (reset) before the `continue` command.
- The `continue` command with *IgnoreCount* specified can be executed on condition that the target program has been executed at least once and is currently halted because break conditions are met. In this case, a break caused by the [Stop] key is not assumed since break conditions are met. If *IgnoreCount* is specified while the target program has never been made to break once, the specification is ignored.
- If the target program has been halted by one cause of a break, and the `continue` command is executed with *IgnoreCount* specified after clearing that break setting, an error is assumed. The same applies when other break conditions have been set.
- If break conditions other than the one that stopped the target program must be ignored a specified number of times, specify break conditions and the number of times that a break hit is to be ignored in the `ignore` command. Then execute the `continue` command without any parameters.

**Operation**

Executes the target program from the current PC address.

A temporary break can be specified at one location, causing the program to stop before executing that breakpoint. A hardware PC break is used for this temporary break, which is cleared when the program breaks once. When a temporary break is specified, assembly sources other than the C source are executed continuously.

If the program does not pass the breakpoint set (a miss), the program runs continuously until made to break by one of the following causes:

- Other set break conditions are met.
- The [Stop] button is clicked.
- Control is returned to a higher level from the current level (within the function).
- There is no assembly source or source information (in which case, only the current instruction is executed).

**Format**

**until** *Breakpoint*

*Breakpoint*: Temporary breakpoint

Can be specified by one of the following:

- Function name
- Source file name:line number, or line number only
- \*Address (decimal, hexadecimal, or symbol)

Conditions:  $0 \leq \text{address} \leq 0\text{xfffff}$

**Usage example**

## ■ Example 1

```
(gdb) until main
main () at main.c:10
```

The target program is run with a temporary break specified by a function name. The program breaks before executing the first C instruction in `main()` (that is expanded to mnemonic). The PC on which the program has stopped displays the start address of that instruction (i.e., address of first mnemonic expanded).

## ■ Example 2

```
(gdb) until main.c:10
main () at main.c:10
```

The target program is run with a temporary break specified by line number. Although the breakpoint here is specified in "source file name:line number" format when the breakpoint is to be set in the C source containing the current PC address, it can be specified by simply using a line number like "until 10". For assembly sources, a source file name is always required. When this command is executed, the program breaks before executing the C instruction on line 10 in `main.c`. The PC on which the program has stopped displays the start address of that instruction (i.e., address of first mnemonic expanded). If no instructions exist on line 10 with actual code (i.e., not expanded to mnemonic), the program breaks at the beginning of the first instruction encountered with actual code thereafter.

## ■ Example 3

```
(gdb) until *0xc0001e
main () at main.c:10
```

The target program is run with a temporary break specified by address. The program breaks before executing the instruction stored at that address location. A symbol can also be used, as shown below.

```
(gdb) until *main
main () at main.c:7
```

Note that adding an asterisk (\*) causes even the function name to be regarded as an address.

## GUI

The `until` command can also be executed using the method described below.

1. In the [Source] window, right-click near the left end of a line where you wish to set a temporary breakpoint. A popup menu will appear. (First, confirm whether there is a breakable instruction on that line.)
2. Choose [Jump to Here] from the popup menu.

The target program will start running from the current PC address and stop at this line.

While the `until` command executes target program, the contents displayed in the [Source] and [Registers] windows are updated in real time. The [Memory] window is not updated.

## Notes

- To run the program from the beginning, execute `c17 rst` (reset) before the `until` command.
- If the location set as a temporary breakpoint is a C source line that does not expand to mnemonic, the program does not break at that line. The program breaks at the address of the first mnemonic executed thereafter.
- No temporary breakpoints can be set on the following lines, because an error is assumed.
  - Extended instruction lines (except for the `ext` instruction at the beginning)
  - Delayed instruction lines (next line after a delayed branch instruction)
- If temporary breakpoints are specified using a nonexistent function name or line number, an error is assumed.
- If temporary breakpoints are specified by an address value that exceeds the 24-bit range, an error is assumed.
- When specifying temporary breakpoints by address value and the address is specified with an odd value, the specified address is adjusted to the 16-bit boundary by assuming `LSB = 0`.

**step** (single-step, every line)**stepi** (single-step, every mnemonic)

[ICD Mini / SIM]

**Operation**

Single-steps the target program from the current PC address. Lines and instructions in the called functions or subroutines also are single-stepped.

**step**: Single-steps the program by executing one source line at a time. In C sources, one line of C instruction (all multiple expanded mnemonics) are executed as one step. In assembly sources, instructions are executed the same way as for **stepi**.

**stepi**: Single-steps the program by executing one assembler instruction (in mnemonic units) at a time.

In addition to one line or instruction, a number of steps to execute can also be specified. However, even before all specified steps are completed, the program may be halted by one of the following causes:

- Already set break conditions are met.
- The [Stop] button is clicked.

**Format****step** [*Count*]**stepi** [*Count*]

*Count*: Number of steps to execute (decimal or hexadecimal)  
One step is assumed if omitted.

Conditions:  $1 \leq \textit{Count} \leq 0x7ffffff$

**Usage example**

## ■ Example 1

`(gdb) step`

The source line displayed on the current PC is executed.

## ■ Example 2

`(gdb) stepi`

The instruction (in mnemonic units) is executed at the address displayed on the current PC.

## ■ Example 3

```
(gdb) step 10
sub (k=5) at main.c:20
```

Ten lines are executed from the source line displayed on the current PC.

## ■ Example 4

```
(gdb) stepi 10
main () at main.c:13
```

Ten instructions (in mnemonic units) are executed from the address displayed on the current PC.

**GUI**

The **step** and **stepi** commands can also be executed using one of the methods below. In such case, however, you cannot specify the number of steps to execute. (Only one step of the program is executed.)

To execute the **step** command

- Click the [Step] button in the [Source] window.



[Step] button

- Choose [Step] from the [Control] menu in the [Source] window.
- Enter [s] ([S]) while the [Source] window is open and displayed in SOURCE mode.

When the program is single-stepped using the method above, information about the line executed appears in the [Console] window and is also output to a log file at log output.

To execute the `stepi` command

- Click the [Step Asm Inst] button in the [Source] window.



[Step Asm Inst] button

- Choose [Step Asm Inst] from the [Control] menu in the [Source] window.
- Enter [s] ([S]) while the [Source] window is open and displayed in other than SOURCE mode.  
When the program is single-stepped using the method above, information about the line executed appears in the [Console] window and is also output to a log file at log output.

While the `step` or `stepi` command executes the target program, the contents displayed in the [Source] and [Registers] windows are updated each time one step is executed. The [Memory] window is not updated.

#### Notes

- The program cannot be single-stepped from an address that does not have source information (i.e., debugging information included in the object). The program can be run continuously, however, by using the `continue` command.
- To run the program from the beginning, execute `c17 rst` (reset) before `step` or `stepi`.
- Even with `stepi`, `ext`-based extended instructions are executed collectively (i.e., entire extended instruction set consisting of two or three instructions) as one step.
- Interrupts are accepted even while single-stepping the program.  
Similarly, the `halt` and `slp` instructions are executed while single-stepping the program, causing the CPU to enter standby status. The CPU exits standby status when an external interrupt is generated. Clicking the [Stop] button also releases the CPU from standby mode.

**next** (single-step with skip, every line)

**nexti** (single-step with skip, every mnemonic)

[ICD Mini / SIM]

#### Operation

Single-steps the target program from the current PC address. The basic operations here are the same as with `step` and `stepi`, except that when a function or subroutine call is encountered, all lines or instructions in the called function or subroutine are executed successively as one step until returning to a higher level.

**next**: Single-steps the program by executing one source line at a time. In C sources, one line of C instruction (all multiple expanded mnemonics) are executed as one step. In assembly sources, instructions are executed the same way as for `nexti`.

**nexti**: Single-steps the program by executing one assembler instruction (in mnemonic units) at a time.

In addition to one line or instruction, a number of steps to execute can also be specified. However, even before all specified steps are completed, the program may be halted by one of the following causes:

- Already set break conditions are met.
- The [Stop] button is clicked.

#### Format

**next** [*Count*]

**nexti** [*Count*]

*Count*: Number of steps to execute (decimal or hexadecimal)  
One step is assumed if omitted.

Conditions:  $1 \leq \textit{Count} \leq 0x7ffffff$

#### Usage example

##### ■ Example 1

```
(gdb) next
```

The source line displayed on the current PC is executed. When the source is a function or subroutine call, the function or subroutine called is also executed until returning to a higher level.

##### ■ Example 2

```
(gdb) nexti
```

The instruction (in mnemonic units) is executed at the address displayed on the current PC. When the instruction is a subroutine call, the subroutine called is also executed until returning to a higher level.

##### ■ Example 3

```
(gdb) next 10
sub (k=5) at main.c:20
```

Ten lines are executed from the source line displayed on the current PC.

##### ■ Example 4

```
(gdb) nexti 10
main () at main.c:13
```

Ten instructions (in mnemonic units) are executed from the address displayed on the current PC.



## GUI

The `next` and `nexti` commands can also be executed using one of the methods below. In such case, however, you cannot specify the number of steps to execute. (Only one step of the program is executed.)

To execute the `next` command

- Click the [Next] button in the [Source] window.



[Next] button

- Choose [Next] from the [Control] menu in the [Source] window.
- Enter [n] ([N]) while the [Source] window is open and displayed in SOURCE mode.

When the program is single-stepped using the method above, information about the line executed appears in the [Console] window and is also output to a log file at log output.

To execute the `nexti` command

- Click the [Next Asm Inst] button in the [Source] window.



[Next Asm Inst] button

- Choose [Next Asm Inst] from the [Control] menu in the [Source] window.
- Enter [n] ([N]) while the [Source] window is open and displayed in other than SOURCE mode.

When the program is single-stepped using the method above, information about the line executed appears in the [Console] window and is also output to a log file at log output.

While the `next` or `nexti` command executes the target program, the contents displayed in the [Source] and [Registers] windows are updated each time one step is executed. The [Memory] window is not updated.

## Notes

- The program cannot be single-stepped from an address that does not have source information (i.e., debugging information included in the object). The program can be run continuously, however, by using the `continue` command.
- To run the program from the beginning, execute `c17 rst` (reset) before `next` or `nexti`.
- Even with `nexti`, `ext`-based extended instructions are executed collectively (i.e., entire extended instruction set consisting of two or three instructions) as one step.
- Interrupts are accepted even while single-stepping the program. Similarly, the `halt` and `slp` instructions are executed while single-stepping the program, causing the CPU to enter standby mode. The CPU exits standby mode when an external interrupt is generated. Clicking the [Stop] button also releases the CPU from standby mode.

**finish** (finish function)

[ICD Mini / SIM]

**Operation**

Executes the target program from the current PC address and causes it stop upon returning from the current function to a higher level. The instruction at the return position is not executed.

Even before a return, however, the program may be halted by one of the following causes:

- Already set break conditions are met.
- The [Stop] button is clicked.

**Format****finish****Usage example**

```
(gdb) finish
```

The target program is executed from the current PC address and halted after a return.

**GUI**

The `finish` command can also be executed using one of the methods below.

- Click the [Finish] button in the [Source] window.



[Finish] button

- Choose [Finish] from the [Control] menu in the [Source] window.
- Enter [f] ([F]) while the [Source] window is open.

While the `finish` command is executing the target program, the contents displayed in the [Source] and [Registers] windows are updated in real time each time one step is executed. The [Memory] window is not updated.

**Notes**

When the `finish` command is executed at the highest level (e.g., boot routine), the program does not stop. If no breaks are set, use the [Stop] button to halt the program.

**c17 callmd** (set user function call mode)

[ICD Mini / SIM]

**Operation**

Sets the destination at which to output execution results after executing the `c17 call` (user function call) command.

**Format**

`c17 callmd Mode [Filename]`

*Mode*: One of the following numeric values that specify result output destination:

- 1 Display in the [Console] window (default).
- 2 Write to a file.
- 3 Display in the [Console] window and write to a file.

*Filename*: Output file name (not effective when *Mode* = 1)

**Usage example**

```
(gdb) c17 callmd 2 call.txt
```

The execution results of the `c17 call` command to be executed are written to file `call.txt`.

**Notes**

When mode 2 (= file) is selected as the output destination, a file is created in the current directory during `c17 call` command execution, with the results written to the file (which is then closed) when the `c17 call` command is completed. If an existing file name is specified, the file is overwritten with the new execution results.

**c17 call** (call user function)

[ICD Mini / SIM]

**Operation**

Calls a user function.  
However, an assembler entry program is required.

**Format**

**c17 call** *Function* [*Arg1* [*Arg2* [*Arg3*]]]

*Function*: The function to be called (function name or decimal/ hexadecimal start address)

*Arg1-3*: Argument (decimal, hexadecimal, or symbol)

Conditions: Up to three arguments can be specified.

**Usage example**

```
(gdb) c17 callmd 1
(gdb) c17 call print_data 1 2 3
arg1=1, arg2=2, arg3=3      (Execution result)
```

The function `print_data()` is called after specifying three arguments.

**User function and entry routine**

When `c17 call` is invoked, **gdb** executes a specified user function and receives a return value from `%r0`. If the return value is -1 (0xfffff), **gdb** terminates the session without performing anything. When the value received is other than that, **gdb** interprets it as the start address of a packet passed from the function and displays internal data of the packet in the [Console] window or directly writes it to a file in its original form. The `c17 callmd` command specifies the output destination. The default output destination is the [Console] window.

The following shows the configuration of the packet returned by a user function.

*data size* (4 bytes) *data* ...

A packet must always start from a 4-byte boundary. A user function must set the start address of this packet in `%r0`. If packets cannot be returned, the user function should set -1 in `%r0`.

The following shows an example of a user function to be called.

The user function must always end with `"jpa %r1"`.

**Example entry program (assembler)**

```
#define SP_INI 0x0800          ; sp is in end of 1KB internal RAM
      jpa      %r1            ; back to mini monitor

;      ****28 - ****3f DSIO command area, 24byte
;
;      ****28          WBUF data1          DSIO output command
;      ****2c          WBUF data2
;
;      ****32          RBUF ID             DSIO input command
;      ****33          RBUF size
;      ****34          RBUF addr
;      ****38          RBUF data1
;      ****3c          RBUF data2

;
; input
;      %r0 : debug ram start address = ****28
;      %r1 : return address( mini Monitor )
; output
;      ****28 WBUF data1 : gdb output message start address
;
;      <message format>
;      size      : message size (4 byte)
;      message   : String data
;
```

```

#define SP_INI 0x1000

.global ext_test
ext_test:
    ld.a    %r7, %sp          ;
    Xld.a   %r2, SP_INI      ; set SP
    ld.a    %sp, %r2
    ld.a    -[%sp], %r7      ; save SP
    ld.a    -[%sp], %r1      ; save return address
    ld.a    -[%sp], %r0      ; save debug ram start address
    ld.a    %r6, %r0         ; %r0 : debug ram start address = ****28
    add.a   %r6, 0x0c
    ld      %r0, [%r6]+      ; Set Arg1
    ld      %r1, [%r6]+
    ld      %r2, [%r6]+      ; Set Arg2
    ld      %r3, [%r6]+
    ld      %r5, [%r6]+      ; Set Arg3
    ld      %r4, [%r6]+
    sub.a   %sp, 4
    ld      [%sp+0x0], %r5
    ld      [%sp+0x2], %r4
    Xcall   iprint_data     ; enter C program
                                ; return = %r0: message buffer address

    add.a   %sp, 4
    ld.a    %r6, %r0         ;
    ld.a    %r0, [%sp]+      ; restore debug ram start address
    ld.a    [%r0], %r6       ; set %r0 ( message address ) to WBUF data1
    ld.a    %r1, [%sp]+      ; restore return address
    ld.a    %r7, [%sp]+      ; restore SP
    ld.a    %sp, %r7
    jpa     %r1              ; back to mini monitor

```

### User function

```

#include "stdio.h"

void *iprint_data( long arg1, long arg2, long arg3 );

struct {
    long size;
    char buf[0x100];
} tmpbuf;

void *iprint_data( long arg1, long arg2, long arg3 )
{
    tmpbuf.size = sprintf(tmpbuf.buf, "arg1=%d,arg2=%d,arg3=%d", arg1, arg2, arg3);
    return (void*)&tmpbuf;
}

```

### Notes

If any function without an entry program is called, **gdb** may run uncontrollably.

## 10.7.6 CPU Reset Commands

### c17 rst (reset)

[ICD Mini / SIM]

#### Operation

Resets the CPU.

As a result, the CPU is reset to its initial state as shown below.

(1) Internal registers of the CPU

r0–r7: 0x000000

pc: Boot address (reset vector in the trap table)

sp: 0xffffc

psr: 0x00 (IL = 000, IE = 0, CVZN = 0000)

(2) The execution counter is cleared to 0.

(3) The [Source] and [Registers] windows reappear.

Because the PC is set to the boot address, the [Source] window redisplay the program beginning with that address. The [Registers] window reappears with the same settings as (1).

#### Format

```
c17 rst
```

#### Usage example

```
(gdb) c17 rst
```

The CPU is reset.

#### GUI

The `c17 rst` command can also be executed using one of the methods below.

- Click the [Reset] button in the [Source] window.



[Reset] button

- Choose [Reset] from the [Reset] menu in the [Source] window.

The above menu command/button executes the command file `\gnu17\reset.gdb`.

Contents of `reset.gdb` at shipment

```
c17 rst
```

The contents may be edited by the user.

#### Notes

- The contents of memory and debugging status of break and trace are not reset.
- When using **gdb** in ICD Mini mode, the bus status and I/O status are retained.

## c17 rstt (reset target)

[ICD Mini]

### Operation

Outputs the reset signal to the reset input pin on the target board.

### Format

```
c17 rstt
```

### Usage example

```
(gdb) c17 rstt
```

The target is reset.

### Notes

- The `c17 rstt` command can only be used in ICD Mini mode.
- To execute this command, a reset input pin is required on the target board.

## 10.7.7 Interrupt Commands

### **c17 int** (interrupt)

[SIM]

#### Operation

Simulates the generation of an interrupt.

When an interrupt number is specified by this command, the specified interrupt is generated at next program startup.

#### Format

**c17 int** [*No Level*]

*No*: Interrupt number (decimal, hexadecimal, or symbol)

*Level*: Interrupt priority level (decimal, hexadecimal, or symbol)

Conditions:  $0 \leq No \leq 0x1f$ ,  $0 \leq Level \leq 7$

#### Usage example

##### ■ Example 1

```
(gdb) c17 int
```

If no parameters are specified, an NMI is generated.

##### ■ Example 2

```
(gdb) c17 int 3 6
```

Any maskable interrupt number and its priority level can be set.

#### Notes

- The **c17 int** command can only be used in simulator mode.
- Make sure the interrupt number is specified from 0 to 31. If this range is exceeded, an error is assumed.
- Make sure the interrupt priority level is specified from 0 to 7. If this range is exceeded, an error is assumed.
- TTBR is effective even in simulator mode.



## c17 intclear (clear interrupt)

[SIM]

### Operation

Simulates canceling interrupts.  
The interrupt specified by the interrupt number is cleared.

### Format

```
c17 intclear [No]
```

*No*: Interrupt number (decimal, hexadecimal, or symbol)

Conditions:  $0 \leq No \leq 0x1f$

### Usage example

```
(gdb) c17 int 3 6  
(gdb) continue  
(gdb) c17 intclear 3
```

Cancel the interrupt of interrupt number 3.

### Notes

- The `c17 intclear` command can only be used in simulator mode.
- Make sure the interrupt number is specified from 0 to 31. If this range is exceeded, an error is assumed.

**c17 int\_load** (load interrupt event file)

[SIM]

**Operation**

Loads an interrupt event file.

When an event condition written in the event file that has been loaded is met during program execution, the designated interrupt occurs.

**Event file format**

```
Address_NMI_INT_Vector_Level_RES          // Comment
Address_NMI_INT_Vector_Level_RES
Address_NMI_INT_Vector_Level_RES
Address_NMI_INT_Vector_Level_RES
:
Address_NMI_INT_Vector_Level_RES
```

*Address*: Address to generate an event (000000–fffffe)

An interrupt occurs when the PC reaches this address.

*NMI*: 1 = NMI request

0 = No NMI request

*INT*: 1 = Interrupt request specified with the *Vector* and *Level* below

0 = No interrupt request

*Vector*: Interrupt vector number (00–1f, valid when *INT* = 1)

*Level*: Interrupt level (0–7, valid when *INT* = 1)

*RES*: 1 = Reset request

0 = No reset request

The order of interrupt priority is *RES* > *NMI* > *Vector*.

A comment (alphanumeric characters) with "/" prefixed can be written on the right of each event line.

**Format**

**c17 int\_load** *Filename*

*Filename*: Interrupt event file name

**Usage example**

```
(gdb) c17 int_load event.txt
```

Loads the interrupt event file event.txt.

**Example of event file**

```
009002_0_1_10_3_0    An interrupt of which the interrupt vector number = 0x10 and interrupt level
                    = 3 occurs when PC = 0x9002.
009f02_0_0_00_0_1    A reset exception occurs when PC = 0x9f02.
004030_1_1_1c_7_0    An NMI occurs when PC = 0x4030. After that an interrupt of which
                    the interrupt vector number = 0x1c and interrupt level = 7 occurs when
                    interrupts are enabled.
004030_0_1_1c_7_1    A reset exception occurs when PC = 0x4030. Although an interrupt vector
                    number (= 0x1c) and an interrupt level (= 7) has been written, no interrupt
                    will occur even when interrupts are enabled because the INT parameter is set
                    to 0.
```

**Notes**

- The `c17 int_load` command can only be used in simulator mode.
- If the `c17 rst` command is executed when an interrupt event file has been loaded, the event sequence is reset so that the events will be generated from the first line again.
- Up to 256 events (event lines) can be written in an interrupt event file.
- Up to 300 characters can be written in an event line.

## 10.7.8 Break Setup Commands

**break** (set software PC break)

**tbreak** (set temporary software PC break)

[ICD Mini / SIM]

### Operation

Sets a software PC breakpoint. This breakpoint can be set at up to 200 locations. If the PC matches the address set during program execution, the program breaks before executing the instruction at that address. A breakpoint can be set using a function name, line number, or address.

The **break** and **tbreak** commands are functionally the same. The following describes the difference:

**break**: The breakpoints set by **break** are not cleared by a break that occurs when the set point is reached during program execution.

**tbreak**: The breakpoints set by **tbreak** are cleared by one occurrence of a break at the set point.

### Format

**break** [*Breakpoint*]

**tbreak** [*Breakpoint*]

*Breakpoint*: Breakpoint

A breakpoint can be specified with one of the following:

- Function name
- Source file name:line number or line number only
- \*Address (decimal, hexadecimal, or symbol)

When omitted, a breakpoint is set at the address displayed on the current PC.

Conditions:  $0 \leq \text{address} \leq 0\text{xfffffe}$

### Usage example

#### ■ Example 1

```
(gdb) break main
Breakpoint 1 at 0xc0001e: file main.c, line 10.
(gdb) continue
Continuing.
```

```
Breakpoint 1, main () at main.c:10
```

A software PC breakpoint is set at the position specified using a function name.

When the target program is run, it breaks before executing the first C instruction (expanded to mnemonic) in `main()`. The PC on which the program has stopped displays the start address of that instruction (i.e., address of first mnemonic expanded).

#### ■ Example 2

```
(gdb) tbreak main.c:10
Breakpoint 1 at 0xc0001e: file main.c, line 10.
```

A temporary software PC breakpoint is set at the position specified with a line number. Although the breakpoint here is specified in "source file name:line number" format when the breakpoint is to be set in the C source containing the current PC address, it can be specified by simply using a line number like "`tbreak 10`". For assembly sources, a source file name is always required.

If no instructions exist on the specified line with actual code (i.e., not expanded to mnemonic), a breakpoint is set at the beginning of the first instruction encountered with actual code thereafter.

When the target program is run, it breaks before executing the C instruction on line 10 in `main.c`. The PC on which the program has stopped displays the start address of that instruction (i.e., address of first mnemonic expanded). If no instructions exist on line 10 with actual code, the program breaks at the beginning of the first instruction encountered with actual code thereafter. Because the breakpoint is set by **tbreak**, it is cleared after a break.

### ■ Example 3

```
(gdb) break *0xc0001e
```

Note: breakpoint 1 also set at pc 0xc0001e.  
Breakpoint 2 at 0xc0001e: file main.c, line 10.

A software PC breakpoint is set at the position specified using an address.

When the target program is run, it breaks before executing the instruction at that address. A symbol can also be used, as shown below.

```
(gdb) tbreak *main
```

Breakpoint 3 at 0xc0001c: file main.c, line 7.

Note that adding an asterisk (\*) causes even a function name to be regarded as an address.

#### Breakpoint management

The breakpoints that you set are sequentially assigned break numbers beginning with 1, regardless of the types of breaks set, and are displayed as a message in the [Console] window when you execute a break setup command. (See the examples above.) These numbers are required to disable/enable or delete breakpoints individually at a later time. Even when you delete breakpoints, the breakpoint numbers are not moved up (to reuse deleted numbers) until after you quit the debugger.

To manipulate the breakpoints you set, use the following commands:

```
disable:           Disables a breakpoint.
enable:           Enables a breakpoint.
delete or clear:   Deletes a breakpoint.
ignore:           Specifies the number of times a break is disabled.
info breakpoints: Displays a list of breakpoints.
```

For details, see the description of each command.

#### GUI

A software PC breakpoint can also be set by following the procedure below.

##### To set a software PC breakpoint

1. Display the source file in which you wish to set a breakpoint in the [Source] window. To set a breakpoint at an address, display the file in ASSEMBLY or MIX mode.
2. The lines at which breakpoints can be set are marked by a bar "-" at the beginning of the line. Move the cursor near the beginning of a line at which you wish to set a breakpoint. The cursor will change shape to a circle (white ○). Click the mouse button there. The "-" mark at the beginning of the line will change to ■ (in red by default), indicating that a breakpoint has been set. Clicking on it again clears the breakpoint you have set.  
Otherwise, right-clicking at the "-" position will display a popup menu, so you can choose [Set Breakpoint] from the menu to set a breakpoint.

##### To set a temporary software PC breakpoint

1. Display the source file in which you wish to set a breakpoint in the [Source] window. To set a breakpoint at an address, display the file in ASSEMBLY or MIX mode.
2. The lines at which breakpoints can be set are marked by a bar "-" at the beginning of the line. Move the cursor near the beginning of a line at which you wish to set a breakpoint. The cursor will change shape to a circle (white ○). Right-click there to display a popup menu, allowing you to choose [Set Temporary Breakpoint]. The "-" mark at the beginning of the line will change to ■ (in orange by default), indicating that a breakpoint has been set. Clicking on it again clears the breakpoint you have set.

The ■ mark (in red or orange) indicates that the software PC breakpoint is effective. The red and orange colors of the mark can be changed to any desired color in the [Source Preferences] dialog box, which appears when choosing [Source...] from the [Preferences] menu in the [Source] window. Any breakpoint disabled by a command, etc. changes color to black.

Breakpoints can be disabled or enabled again in the [Breakpoints] window. For details, see Section 10.4.6, "[Breakpoints] Window".

## Notes

- Software PC breakpoints can be set at up to 200 locations. If this limit is exceeded, an error is assumed. Note that this break count includes the software PC breakpoints used by the debugger in other functions.
- C source lines that are not expanded to mnemonic cannot be specified as a location at which to set a software PC breakpoint. Specifying such a C line sets a software PC breakpoint at the address of the first instruction to be executed next.
- When a function name or the beginning C source line in a function is specified as the position where to set a software PC breakpoint, the program execution will break at the start address of the first C source (i.e., instruction to be expanded to mnemonic) in the function. Although a `ld` instruction to save register contents is inserted at the beginning of the function during compilation, this instruction is executed before the program breaks. To make the program break before executing this instruction, specify a software PC breakpoint using the address value of that instruction.
- No software PC breakpoints can be set at the following lines.
  - Extended instruction lines (except for the `ext` instruction at the beginning)
  - Delayed instruction lines (next line after a delayed branch instruction)
- If software PC breakpoints are specified using a nonexistent function name or line number, an error is assumed.
- When specifying a software PC breakpoint by an address value, the specified address will be adjusted to the 16-bit boundary by assuming `LSB = 0` if it is an odd value. Furthermore, an error occurs if the specified address exceeds the 24-bit range.
- Software PC breaks are implemented by an embedded `brk` instruction and therefore cannot be used for target board ROM in which instructions cannot be embedded. In such case, use hardware PC breaks instead.

**hbreak** (set hardware PC break)**thbreak** (set temporary hardware PC break)

[ICD Mini / SIM]

**Operation**

Sets a hardware PC breakpoint. This breakpoint can be set at only one location. When the PC matches the address set during program execution, the program breaks before executing the instruction at that address. A breakpoint can be set using a function name, line number, or address.

The **hbreak** and **thbreak** commands are functionally the same. The following describes the difference:

**hbreak:** The breakpoints set by **hbreak** are not cleared by a break that occurs when the set point is reached during program execution.

**thbreak:** The breakpoints set by **thbreak** are cleared by one occurrence of a break at the set point.

**Format**

**hbreak** [*Breakpoint*]

**thbreak** [*Breakpoint*]

*Breakpoint:* Breakpoint

A breakpoint can be specified with one of the following:

- Function name
- Source file name:line number or line number only
- \*Address (decimal, hexadecimal, or symbol)

When omitted, a breakpoint is set at the address displayed on the current PC.

Conditions:  $0 \leq \text{address} \leq 0\text{xfffffe}$

**Usage example**

## ■ Example 1

```
(gdb) hbreak main
```

```
Hardware assisted breakpoint 1 at 0xc0001e: file main.c, line 10.
```

```
(gdb) continue
```

```
Continuing.
```

```
Breakpoint 1, main () at main.c:10
```

A hardware PC breakpoint is set at the position specified using a function name.

When the target program is run, it breaks before executing the first C instruction (expanded to mnemonic) in `main()`. The PC on which the program has stopped displays the start address of that instruction (i.e., address of first mnemonic expanded).

## ■ Example 2

```
(gdb) thbreak main.c:10
```

```
Hardware assisted breakpoint 1 at 0xc0001e: file main.c, line 10.
```

A temporary hardware PC breakpoint is set at the position specified with a line number. Although the breakpoint here is specified in "source file name:line number" format when the breakpoint is to be set in the C source containing the current PC address, it can be specified by simply using a line number like "thbreak 10". For assembly sources, a source file name is always required.

If no instructions exist on the specified line with actual code (i.e., not expanded to mnemonic), a breakpoint is set at the beginning of the first instruction encountered with actual code thereafter.

When the target program is run, it breaks before executing the C instruction line 10 in `main.c`. The PC on which the program has stopped displays the start address of that instruction (i.e., address of first mnemonic expanded). If no instructions exist on line 10 with actual code, the program breaks at the beginning of the first instruction encountered with actual code thereafter. Because the breakpoint is set by **thbreak**, it is cleared after a break.

### ■ Example 3

```
(gdb) hbreak *0xc0001e
```

Note: breakpoint 1 also set at pc 0xc0001e.

Hardware assisted breakpoint 2 at 0xc0001e: file main.c, line 10.

A hardware PC breakpoint is set at the position specified using an address.

When the target program is run, it breaks before executing the instruction at that address. A symbol can also be used, as shown below.

```
(gdb) thbreak *main
```

Hardware assisted breakpoint 3 at 0xc0001c: file main.c, line 7.

Note that adding an asterisk (\*) causes even a function name to be regarded as an address.

#### Breakpoint management

The breakpoints you set are sequentially assigned break numbers beginning with 1, regardless of which types of breaks you set, and are displayed as a message in the [Console] window when you execute a break setup command. (See the examples above.) These numbers are required when you disable/enable or delete breakpoints individually at a later time. Even when you delete breakpoints, the breakpoint numbers are not moved up (to reuse deleted numbers) until after you quit the debugger.

To manipulate the breakpoints you set, use the following commands:

disable:	Disables a breakpoint.
enable:	Enables a breakpoint.
delete or clear:	Deletes a breakpoint.
ignore:	Specifies the number of times a break is disabled.
info breakpoints:	Displays a list of breakpoints.

For details, see the description of each command.

#### GUI

A hardware PC breakpoint can also be set by following the procedure below.

1. Display the source file in which you wish to set a breakpoint in the [Source] window. To set a breakpoint at an address, display the file in ASSEMBLY or MIX mode.
2. The lines at which breakpoints can be set are marked by a bar "-" at the beginning of the line. Move the cursor near the beginning of a line at which you wish to set a breakpoint. The cursor will change shape to a circle (white ○). Right-click there to display a popup menu, allowing you to choose [Set Hardware Breakpoint]. The "-" mark at the beginning of the line will change to ■ (in blue by default), indicating that a breakpoint has been set. Clicking on it again clears the breakpoint you have set.

The ■ mark (in blue) indicates that the hardware PC breakpoint is effective. The blue color of the mark can be changed to any desired color in the [Source Preferences] dialog box, which appears when choosing [Source...] from the [Preferences] menu in the [Source] window. Any breakpoint disabled by a command, etc. changes color to black.

Breakpoints can be disabled or enabled again in the [Breakpoints] window. For details, see Section 10.4.6, "[Breakpoints] Window".

The ■ mark (in orange) indicates that the temporary hardware PC breakpoint is effective.

### Notes

- A valid hardware PC breakpoint can be set at only one location. The debugger allows you to specify two or more hardware PC breakpoints, note, however, that they will be set as disabled breakpoints. Be aware that a temporary hardware PC breakpoint is included in this break count.
- C source lines that are not expanded to mnemonic cannot be specified as a location where to set a hardware PC breakpoint. Specifying such a C line sets a hardware PC breakpoint at the address of the first instruction to be executed next.
- If a function name or the beginning C source line in a function is specified as the position at which to set a hardware PC breakpoint, the program execution will break at the start address of the first C source (i.e., instruction to be expanded to mnemonic) in the function. Although a `ld` instruction to save registers is inserted at the beginning of the function during compilation, this instruction is executed before the program breaks. To make the program break before executing this instruction, specify a breakpoint with the address value of that instruction.
- No hardware PC breakpoints can be set at the following lines, because an error is assumed and the target program can no longer be executed. (This problem may be resolved, however, by clearing the breakpoint.)
  - Extended instruction lines (except for the `ext` instruction at the beginning)
  - Delayed instruction lines (next line after a delayed branch instruction)
- If hardware PC breakpoints are specified using a nonexistent function name or line number, an error is assumed.
- When specifying hardware PC breakpoints by address value and the address is specified with an odd value, the specified address is adjusted to the 16-bit boundary by assuming  $LSB = 0$ .



**delete** (clear break by break number)

[ICD Mini / SIM]

**Operation**

Deletes all breakpoints currently set or one or more breakpoints individually by specifying a break number.

**Format**

**delete** [*BreakNo*]

*BreakNo*: Break number (decimal or hexadecimal)

When this entry is omitted, all breakpoints are deleted.

**Usage example**

```
(gdb) info breakpoints      (displays a breakpoint list.)
Num Type          Disp Enb Address      What
1  breakpoint     keep y  0x00c00038 in sub at main.c:20
2  breakpoint     keep y  0x00c00030 in main at main.c:14
3  breakpoint     keep y  0x00c0003c in sub at main.c:22
```

Let's assume that breakpoints have been set as shown above.

**Example 1**

```
(gdb) delete 1 2
(gdb) info breakpoints
Num Type          Disp Enb Address      What
3  breakpoint     keep y  0x00c0003c in sub at main.c:22
```

When you specify a break number, only that break can be cleared. You can specify multiple break numbers at a time.

**Example 2**

```
(gdb) delete
(gdb) info breakpoints
No breakpoints or watchpoints.
```

When a break number is omitted, all breakpoints are cleared. To avoid inadvertent deletion, a dialog box is displayed for your confirmation. When "Delete all breakpoints?" appears, respond by clicking the [Yes] or [No] button. Choosing [Yes] clears all breaks; choosing [No] cancels deletion.

**GUI**

Normal software PC breakpoints, temporary software PC breakpoints, and hardware PC breakpoints can be deleted using one of the methods below. Temporary hardware PC breakpoints cannot be deleted.

**Deletion method 1**

Display the source containing the set breakpoint you wish to delete in the [Source] window. The source lines with set breakpoints are marked by ■ at the beginning (red for normal software PC breakpoints, orange for temporary software PC breakpoints, blue for hardware PC breakpoints, or black for disabled PC breakpoints). Click on ■ to delete the breakpoint there. The mark at the beginning of that source line will change to "-". Otherwise, you can right-click at the ■ position and choose [Delete breakpoint] from the ensuing popup menu to delete the breakpoint.

- Notes:**
- If you set a breakpoint by specifying an address, it is possible that multiple breakpoints (multiple expanded mnemonics) have been set on one C source line. Even if you click on ■ at the beginning of the source line while the [Source] window is displayed in SOURCE mode, the mark does not change to "-". To delete all breakpoints set on a given source line, you must click as many times as the number of breakpoints there. At this time, the breakpoints are deleted in order of ascending addresses. For display in ASSEMBLY or MIX mode, you can confirm each breakpoint individually as you delete them.
  - The color of mark ■ representing a breakpoint (red, orange, or blue by default) can be changed to any desired color in the [Source Preferences] dialog box that appears by choosing [Source...] from the [Preferences] menu in the [Source] window.

### Deletion method 2

This method is employed in the [Breakpoints] window. Choose the breakpoint you wish to delete by clicking it in the list in the window, then choose [Remove] from the [Breakpoint] menu. Otherwise, you can choose [Remove All] from the [Global] menu to delete all breakpoints. For details, see Section 10.4.6, "[Breakpoints] Window".

#### Notes

- Break numbers are sequentially assigned to each breakpoint you set, beginning with 1. If you do not know the break number of a breakpoint you wish to delete, use the `info breakpoints` command to confirm as in the example above.
- The `delete` command clears all break settings. To disable a breakpoint temporarily, use the `disable` or `ignore` command.
- Note that specifying a break number not set displays the "No breakpoint number N." message, with no breakpoints being deleted.

**clear** (clear break by break position)

[ICD Mini / SIM]

**Operation**

Deletes PC breakpoints (including temporary breakpoints) currently set individually by specifying a set position (function name, line number, or address).

**Format**

**clear** [*Breakpoint*]

*Breakpoint*: Breakpoint

Can be specified by one of the following:

- Function name
- Source file name:line number or line number only
- \*Address (decimal, hexadecimal, or symbol)

When this entry is omitted, all breakpoints set in the source that includes the current PC address are deleted.

Conditions:  $0 \leq \text{address} \leq 0\text{xfffffe}$

**Usage example**

```
(gdb) info breakpoints      (displays a breakpoint list.)
Num Type           Disp Enb Address      What
1  breakpoint      keep y  0x00c0001e in main at main.c:10
2  breakpoint      keep y  0x00c00038 in sub at main.c:20
3  breakpoint      keep y  0x00c0003c in sub at main.c:22
4  breakpoint      keep y  0x00c00042 in sub at main.c:22
```

Let's assume that breakpoints have been set as shown above. Although break numbers 3 and 4 are at different addresses, the breakpoints are set on one line in terms of the C source. (This applies when breakpoints are set at addresses displayed in ASSEMBLY mode.)

**Example 1**

```
(gdb) clear main.c:22
Deleted breakpoints 4 3
(gdb) info breakpoints
Num Type           Disp Enb Address      What
1  breakpoint      keep y  0x00c0001e in main at main.c:10
2  breakpoint      keep y  0x00c00038 in sub at main.c:20
```

When you specify a line number, all breakpoints set on the source line are cleared.

**Example 2**

```
(gdb) clear main
Deleted breakpoint 1
(gdb) info breakpoints
Num Type           Disp Enb Address      What
2  breakpoint      keep y  0x00c00038 in sub at main.c:20
```

When you specify a function name, the breakpoint set in the first C instruction within the function (expanded to mnemonic) is cleared. Use this method to delete breakpoints that have been set by "break function name", etc.

**Example 3**

```
(gdb) clear
Deleted breakpoint 2
(gdb) info breakpoints
No breakpoints or watchpoints.
```

When parameters are omitted, all breakpoints set in the source that includes the address displayed on the current PC (main.c in this example) are deleted. If no breakpoints are set in the source, the "No source file specified." message appears, with no breakpoints being deleted.

## GUI

Normal software PC breakpoints, temporary software PC breakpoints, and hardware PC breakpoints can be deleted using one of the methods below. Temporary hardware PC breakpoints cannot be deleted.

## Deletion method 1

Display the source containing the set breakpoint you wish to delete in the [Source] window. The source lines with set breakpoints are marked by ■ at the beginning (red for normal software PC breakpoints, orange for temporary software PC breakpoints, blue for hardware PC breakpoints or black for disabled PC breakpoints).

Click on ■ to delete the breakpoint there. The mark at the beginning of that source line will change to "-". Otherwise, you can right-click at the ■ position and choose [Delete breakpoint] from the ensuing popup menu to delete the breakpoint.

- Notes:**
- If you set a breakpoint by specifying an address, it is possible that multiple breakpoints (multiple expanded mnemonics) have been set on one C source line. Even if you click on ■ at the beginning of the source line while the [Source] window is displayed in SOURCE mode, the mark does not change to "-". To delete all breakpoints set on a given source line, you must click as many times as the number of breakpoints there. At this time, the breakpoints are deleted in order of ascending addresses. For display in ASSEMBLY or MIX mode, you can confirm each breakpoint individually as you delete them.
  - The color of mark ■ representing a breakpoint (red, orange, or blue by default) can be changed to any desired color in the [Source Preferences] dialog box that appears by choosing [Source...] from the [Preferences] menu in the [Source] window.

## Deletion method 2

This method is employed in the [Breakpoints] window. Choose the breakpoint you wish to delete by clicking it in the list in the window, then choose [Remove] from the [Breakpoint] menu. Otherwise, you can choose [Remove All] from the [Global] menu to delete all breakpoints. For details, see Section 10.4.6, "[Breakpoints] Window".

## Notes

- The `clear` command completely clears break settings. To disable a breakpoint temporarily, use the `disable` or `ignore` command.
- If you specify a function name, line number, or address for which no breakpoints are set, an error is assumed.

**enable** (enable breakpoint)**disable** (disable breakpoint)

[ICD Mini / SIM]

**Operation****enable:** Enables a currently disabled breakpoint to make it effective again.**disable:** Disables a currently effective breakpoint to make it ineffective.

Breakpoints are effective when set by a break command and remain effective. The `disable` command disables these breakpoints without deleting them. Once disabled, the breakpoints are ineffective and the program does not break until said breakpoints are reenabled by the `enable` command.

**Format****enable** [*BreakNo*]**disable** [*BreakNo*]*BreakNo:* Break number (decimal or hexadecimal)

When this entry is omitted, all breakpoints are disabled or enabled.

**Usage example**

```
(gdb) info breakpoints      (displays a breakpoint list.)
Num Type          Disp Enb Address    What
1  breakpoint     keep y  0x00c0001c in main at main.c:7
2  breakpoint     keep y  0x00c0001e in main at main.c:10
3  breakpoint     keep y  0x00c00028 in main at main.c:13
4  breakpoint     keep y  0x00c00038 in sub at main.c:20
```

Let's assume that breakpoints have been set as shown above. The effective breakpoints are marked by 'y' in the Enb column.

**Example 1**

```
(gdb) disable 1 3
(gdb) info breakpoints
Num Type          Disp Enb Address    What
1  breakpoint     keep n  0x00c0001c in main at main.c:7
2  breakpoint     keep y  0x00c0001e in main at main.c:10
3  breakpoint     keep n  0x00c00028 in main at main.c:13
4  breakpoint     keep y  0x00c00038 in sub at main.c:20
```

When executing the `disable` command with a break number attached, note that only the specified break is disabled. You can specify multiple break numbers at a time. Ineffective breakpoints are marked by 'n' in the Enb column.

**Example 2**

```
(gdb) enable
(gdb) info breakpoints
Num Type          Disp Enb Address    What
1  breakpoint     keep y  0x00c0001c in main at main.c:7
2  breakpoint     keep y  0x00c0001e in main at main.c:10
3  breakpoint     keep y  0x00c00028 in main at main.c:13
4  breakpoint     keep y  0x00c00038 in sub at main.c:20
```

When a break number is omitted, all breakpoints are enabled (or disabled) simultaneously.

## GUI

Normal software PC breakpoints, temporary software PC breakpoints, and hardware PC breakpoints can be disabled or enabled by using one of the methods below. Temporary hardware PC breakpoints cannot be switched between enabled and disabled states.

## Switch method 1

Display the source containing the set breakpoint you wish to disable or enable in the [Source] window. The source lines with set breakpoints are marked by ■ at the beginning (red for normal software PC breakpoints, orange for temporary software PC breakpoints, blue for hardware PC breakpoints or black for disabled PC breakpoints).

To disable a breakpoint, right-click on the source line marked by ■ (red, orange, or blue) and choose [Disable breakpoint] from the ensuing popup menu. The mark will change to ■ (black), indicating that the breakpoint has been disabled.

To enable a breakpoint, right-click on the source line marked by ■ (black) and choose [Enable breakpoint] from the ensuing popup menu. The mark will change to ■ (red, orange, or blue), indicating that the breakpoint has been enabled.

- Notes:**
- If you set a breakpoint by specifying an address, it is possible that multiple breakpoints have been set on one C source line (multiple expanded mnemonics). If you perform the operation above on such a source line while the [Source] window is displayed in SOURCE mode, all breakpoints set on that source line will be disabled (when effective) or enabled (when ineffective). For display in ASSEMBLY or MIX mode, you can confirm each breakpoint individually as you disable or enable the breakpoint.
  - The color of mark ■ representing a breakpoint (red, orange, or blue by default) can be changed to any desired color in the [Source Preferences] dialog box that appears by choosing [Source...] from the [Preferences] menu in the [Source] window.

## Switch method 2

This method is employed in the [Breakpoints] window. Check the break list in the window to confirm whether the check box before each item of break information is selected (flagged by a check mark for an effective break) or deselected (with no check mark for an ineffective break). Click the check box for the desired breakpoint to disable an effective breakpoint or enable an ineffective one.

Otherwise, you can choose a breakpoint to disable or enable by clicking on it in the list in the window, then choose [Disable] or [Enable] from the [Breakpoint] menu. You can also choose [Enable All] or [Disable All] from the [Global] menu to change the disabled/enabled states of all breakpoints simultaneously. For details, see Section 10.4.6, "[Breakpoints] Window".

## Notes

- Break numbers are sequentially assigned to each breakpoint when set, beginning with 1. If you do not know the break number of a breakpoint you wish to disable or enable, use the `info breakpoints` command to confirm as in the example above.
- The number of breakpoints that can be set is limited. Use the `delete` command to delete unnecessary breakpoints.
- Note that specifying a break number not set displays the "No breakpoint number N." message, with no breakpoints being disabled or enabled.

**ignore** (disable breakpoint with ignore counts)

[ICD Mini / SIM]

**Operation**

Disables a specific break the number of times specified by a break hit count.

**Format**

**ignore** *BreakNo* *Count*

*BreakNo*: Break number (decimal)

*Count*: Number of break hits to be disabled (decimal or hexadecimal)

**Usage example**

```
(gdb) info breakpoints
Num Type          Disp Enb Address      What
1  breakpoint      keep y  0x00c0003c  in sub at main.c:22
2  breakpoint      keep y  0x00c00030  in main at main.c:14
```

```
(gdb) ignore 2 2
Will ignore next 2 crossings of breakpoint 2.
```

Break number 2 is disabled twice.

```
(gdb) continue
Continuing.
```

```
Breakpoint 1, sub (k=1) at main.c:22
(gdb) continue
Continuing.
```

```
Breakpoint 1, sub (k=1) at main.c:22
(gdb) continue
Continuing.
```

```
Breakpoint 2, main () at main.c:14
```

Although the target program passes through the breakpoint twice as it is run twice (by `continue`) above, no break occurs. A break occurs when running the program a third time because the breakpoint is reenabled.

**Notes**

- Break numbers are sequentially assigned to each breakpoint when set, beginning with 1. If you do not know the break number of a breakpoint you wish to disable, use the `info breakpoints` command to confirm as in the example above.
- Count is used to count the number of times a specific break is hit, and not the number of times the target program is run. The count is not decremented unless the program passes through a specified breakpoint.
- The `ignore` command cannot be used to collectively disable multiple breakpoints.
- Note that specifying a break number not set displays the "No breakpoint number N." message, with program execution being aborted.

**info breakpoints** (display breakpoint list)

[ICD Mini / SIM]

**Operation**

Displays a list of breakpoints currently set.

**Format**

**info breakpoints**

**Display**

The breakpoint list is displayed as shown below.

```
(gdb) info breakpoints
Num Type           Disp Enb Address      What
1  breakpoint      keep y  0x00c00026 in main at main.c:11
   breakpoint already hit 1 time
   ignore next 10 hits
2  hw breakpoint  del  n  0x00c00038 in sub at main.c:20
```

Num: Indicates a break number.

Type: Indicates the type of breakpoint.

breakpoint Software PC breakpoint

hw breakpoint Hardware PC breakpoint

Disp: Indicates breakpoint status after a break hit.

keep The breakpoint will not be deleted.

del The breakpoint will be deleted. This means that the breakpoint is a temporary break.

Enb: Indicates whether the breakpoint is effective or ineffective.

y Effective

n Ineffective

Address: Indicates the address at which a breakpoint is set (in hexadecimal).

What: Indicates the location where a breakpoint is set. This information is displayed in "*in function name at source file name:line number*" format.

Moreover, the number of times a breakpoint has thus been hit is displayed in "breakpoint already hit N times" format; the set content of the ignore command is displayed in "ignore next N hits" format.

When breakpoints are not set at any location, the list is displayed as shown below.

```
(gdb) info breakpoints
No breakpoints or watchpoints.
```

**GUI**

A list of software PC breakpoints (including temporary breaks) is displayed in the [Breakpoints] window. For details, see Section 10.4.6, "[Breakpoints] Window".



**c17 timebrk** (set lapse of time break)

[ICD Mini]

**Operation**

Sets a time interval until the program execution is made to break forcibly after it starts. This command also disables this break function.

**Format**

**c17 timebrk *Timer***

*Timer*: Time until program execution is made to break from start in millisecond units (decimal or hexadecimal)

Can be set within the range from 1 to 300000 (milliseconds).

The break function is disabled if 0 is specified. (Default setting at starting up of the debugger)

**Usage example**■ **Example 1**

```
(gdb) c17 timebrk 1000
      timer break on. [1000 ms]
(gdb) cont
Continuing.
```

A forcible break will occur after 1 second from starting the program execution.

■ **Example 2**

```
(gdb) c17 timebrk 0
      timer break off.
```

The lapse of time break is disabled.

**Notes**

- This command cannot be used in simulator mode.
- When a break time has been set once, the lapse of time break is effective until it is disabled with "c17 timebrk 0". A break will occur after the set time has elapsed every time the program is started.
- If another break condition is met or the [Stop] button is clicked before the set time has elapsed, the program being executed is made to break at that point immediately.

## 10.7.9 Symbol Information Display Commands

**info locals** (display local symbol)

**info var** (display global symbol)

[ICD Mini / SIM]

### Operation

Displays a list of symbols.

**info locals:** Displays a list of local variables defined in the current function.

**info var:** Displays a list of global and static variables.

### Format

**info locals**

**info var**

### Usage example

#### ■ Example 1

```
(gdb) info locals
i = 0
j = 2
```

All local symbols defined in the function that includes the current PC address are displayed along with symbol content.

#### ■ Example 2

```
(gdb) info var
All defined variables:

File main.c:
int i;

Non-debugging symbols:
0x00000000 __START_bss
0x00000004 __END_bss
0x00000004 __END_data
0x00000004 __START_data
```

All defined global and static variables are displayed in list form separately for each source file. Displayed under the heading "Non-debugging symbols:" are such global symbols as section symbols defined in other than the source file.

### GUI

A list of local symbols is displayed in the [Local Variables] window. For details, see Section 10.4.8, "[Local Variables] Window".

### Notes

If the current position indicated by the PC address is outside the function (stack frame) (e.g., in boot routine of an assembly source), local symbols are not displayed.

```
(gdb) info locals
No frame selected.
```

**print** (alter symbol value)

[ICD Mini / SIM]

**Operation**

Alters the value of a symbol.

**Format**

**print** *Symbol* [=*Value*]

*Symbol*: Variable name

*Value*: Value used to alter (decimal, hexadecimal, or symbol)

When this entry is omitted, the current symbol value is displayed.

Conditions:  $0 \leq \textit{Value} \leq$  valid range of type

**Usage example**■ **Example 1**

```
(gdb) info local
j = 0
(gdb) print j
$1 = 0
```

When you specify only a variable name, the value of that variable is displayed. The  $\$N$  is a number used to reference this value at a later time. The contents displayed here can be referenced using `print $1`.

■ **Example 2**

```
(gdb) print j=5
$2 = 5
(gdb) info local
j = 5
```

Note that specifying a value changes the variable value to that specified.

**GUI**

The values of local symbols can be altered in the [Local Variables] window.

1. Choose a symbol whose value you wish to alter from the symbol list in the [Local Variables] window, then choose [Edit] from the [Variable] menu or double-click the symbol value displayed.
2. The symbol value is placed in edit mode, so enter a new value for the symbol.

For details, see Section 10.4.8, "[Local Variables] Window".

**Notes**

- If you specify an undefined symbol, an error is assumed.
- Even if the value you have specified exceeds the range of values for the type of variable you wish to alter, no errors are assumed. Only a finite number of low-order bits equivalent to the size of the variable are effective, with excessive bits being ignored. For example, specifying 0x10000 for variable `int` is processed as 0x0000.

## 10.7.10 File Loading Commands

**file** (load debugging information)

[ICD Mini / SIM]

### Operation

Loads only debugging information from elf format object files.  
Use the `load` command to load necessary object code.

### Format

**file** *Filename*

*Filename*: Name of object file in elf format to be debugged (with path also specifiable)

### Usage example

```
(gdb) file sample.elf
```

Debugging information is loaded from `sample.elf` in the current directory.

### GUI

You can also choose [Open...] from the [File] menu in the [Source] window and open an elf format file from the file select dialog box that appears. Selecting a file executes the `file` command. [Source] window display is updated when the source file indicated in loaded debugging information is loaded.

### Notes

- The `file` command only loads debugging information; it does not load object code. Therefore, except when the program is written to target ROM, you cannot start debugging by simply executing the `file` command.
- The `file` command must be executed before the `target` and `load` commands. The following shows the basic sequence of command execution:
 

```
(gdb) c17 rpf sample.par (sets map information.)
(gdb) file sample.elf (this command)
(gdb) target sim (connects the target.)
(gdb) load (loads the program.)
(gdb) c17 rst (resets the CPU.)
```
- Unless executed for elf object files in executable format (generated by the linker), the `file` command results in an error and no files can be loaded. If the loaded file contains no debugging information, an error also results.
- The elf format object files contain information on source files (including the directory structure). For this reason, unless the source files exist in a specified directory in the object file as viewed from the current directory, the source files cannot be loaded. Basically, the series of operations from compiling to debugging should be performed in the same directory.
- Once the `file` command is executed, operation cannot be aborted until the debugger finishes loading the file.

**load** (load program)

[ICD Mini / SIM]

**Operation**

Loads the program and data from elf format object files into target memory.

**Format**

**load** [*Filename*]

*Filename*: Name of object file in elf or Motorola S3 format to be debugged (with path also specifiable)

When this entry is omitted, the file specified previously by the `file` command is loaded. This specification is usually omitted.

**Usage example**

```
(gdb) file sample.elf
(gdb) target sim
(gdb) load
```

The program and data are loaded from `sample.elf` in the current directory (specified by the `file` command) into target memory (computer memory in this example because the debugger operates in simulator mode).

**Notes**

- The `load` command must be executed after the `file` and `target` commands. The following shows the basic sequence of command execution:

```
(gdb) c17 rpf sample.par (sets map information.)
(gdb) file sample.elf    (loads debugging information.)
(gdb) target sim        (connects the target.)
(gdb) load               (this command)
(gdb) c17 rst           (resets the CPU.)
```

- The `load` command loads only several areas of an object file containing the code and data. All other areas are left intact in the previous state before `load` command execution.
- The size of file that can be downloaded for debugging is a maximum of 1,024 bytes per block (`set download-write-size 1024`). Setting a greater value will cause the debugger to operate erratically.

**c17 loadmd** (set program load mode)

[ICD Mini]

**Operation**

Set the mode to load program/data into the target memory from an elf format object file when the `load` command is executed.

**Format**

**c17 loadmd Mode**

*Mode*: Transfer mode

- 0 High-speed byte transfer mode (default)
- 1 Low-speed byte transfer mode

**Usage example**

```
(gdb) file sample.elf
(gdb) target icd usb
(gdb) c17 loadmd 0
(gdb) load
```

The program and data are loaded from `sample.elf` located in the current directory (specified by the `file` command) into the target memory using high-speed 8-bit instructions.

**Notes**

This command cannot be used in simulator mode.

## 10.7.11 Map Information Commands

### c17 rpf (set map information)

[SIM]

#### Operation

Sets memory map information by loading a specified parameter file.

#### Format

**c17 rpf** *Filename*

*Filename*: Name of parameter file (with path also specifiable)

#### Usage example

```
(gdb) c17 rpf sample.par
```

Memory map information is loaded from `sample.par` in the current directory.

#### Notes

- The `c17 rpf` command can only be used in simulator mode.
- For details about parameter files, see Section 10.8, "Parameter Files".
- Make sure the `c17 rpf` command is executed only once prior to the `target` command. The following shows the basic sequence of command execution:

```
(gdb) c17 rpf sample.par (this command)
(gdb) file sample.elf    (loads debugging information.)
(gdb) target sim         (connects the target.)
(gdb) load               (loads the program.)
(gdb) c17 rst            (resets the CPU.)
```

To reset memory map information, quit the debugger temporary, then restart it and execute the `c17 rpf` command.

- The debugger allocates a PC memory area according to the memory map information in the parameter file loaded. If a required area cannot be allocated (with error message "Cannot allocate memory." displayed), correct the area size in the parameter file by making it smaller.
- No software PC breaks and temporary software PC breaks can be set in areas assigned the "ROM" attribute in a parameter file (`*.par`).
- It is not necessary to execute this command in ICD Mini mode. This command should be executed in simulator mode.

If this command is not executed in simulator mode, the memory map information is set as follows:

```
TTBR  0x8000
RAM    0x0-0xfffff, R/W wait cycle = 0, 16-bit access, little endian
```

**c17 map** (display map information)

[SIM]

**Operation**

Displays memory map information set by a parameter file.

**Format**

**c17 map**

**Usage example**

```
(gdb) c17 map
CPU : C17
Memory map information Type  Wait(r/w) Size  Endian
      00008000          TTBR
      00000000-00ffffff  RAM    0/0    16Bit Little
      00000000-00003eff  STACK
```

**CPU:** Indicates the type of CPU.

C17: S1C17 Core

**Memory map information:**

Indicates the range of memory addresses to which the device is allocated (start address–end address of the area) in hexadecimal.

**Type:** Indicates the type of memory or device.

**Wait (r/w):** Indicates the number of wait cycles inserted during read/write operation.

**Size:** Indicates the data width of the device (in bits).

**Endian:** Indicates the endian format (little or big endian) of the area.

When memory map information has yet to be loaded by the `c17 rpf` command, a message appears like the one shown below.

```
(gdb) c17 map
CPU : C17
Memory map information Type  Wait(r/w) Size  Endian
      00100000          TTBR
      00000000-00ffffff  RAM    0/0    16Bit Little
```

**Notes**

The `c17 map` command can only be used in simulator mode.



## 10.7.12 Flash Memory Manipulation Commands

### c17 fls (set flash memory)

[ICD Mini]

#### Operation

Sets up flash memory of the target system as required to write data to it.

#### Format

**c17 fls *StartAddr EndAddr ErasePrg WritePrg***

*StartAddr*: Start address of flash memory (decimal, hexadecimal, or symbol)

*EndAddr*: End address of flash memory (decimal, hexadecimal, or symbol)

*ErasePrg*: Start address of erase program (decimal, hexadecimal, or symbol)

*WritePrg*: Start address of write program (decimal, hexadecimal, or symbol)

Conditions:  $0 \leq \textit{StartAddr} \leq \textit{EndAddr} \leq 0\text{xfffff}$ ,  $0 \leq \textit{ErasePrg} \leq 0\text{xfffff}$ ,  $0 \leq \textit{WritePrg} \leq 0\text{xfffff}$

#### Usage example

```
(gdb) c17 fls 0x200000 0x2fffff FLASH_ERASE FLASH_LOAD
```

```
Flash start address = 0x200000, Flash end address = 0x2fffff
```

```
Flash erase routine address = 0x100, Flash write routine address = 0x200 .....done
```

The flash memory area in the target system (0x200000 to 0x2fffff), and addresses of the erase and write routines are set.

#### Notes

- This command cannot be used in simulator mode.
- Before you can erase flash memory of the target system or write data to flash memory, you must have written the data write and erase programs to the specified address locations.

**c17 fle** (erase flash memory)

[ICD Mini]

**Operation**

Erases the contents of flash memory of the target system.

**Format**

**c17 fle** *ControlReg StartBlock EndBlock* [*Timer*]

*ControlReg*: Start address set by **c17 fls** (decimal, hexadecimal, or symbol)

*StartBlock*: First block in erase range (decimal, hexadecimal, or symbol)

*EndBlock*: Last block in erase range (decimal, hexadecimal, or symbol)

When *StartBlock* = *EndBlock* = 0, the entire area is erased.

*Timer*: Timeout value (decimal or hexadecimal)

Specify a time in second. When omitted, a timeout occur in 150 seconds.

**Usage example**

```
(gdb) c17 fle 0x200000 0 0
```

Control Register = 0x200000, Start block = 0x0, End block = 0x0 .....Finish with 0x00000000

The entire area of flash memory is erased.

**Notes**

- This command cannot be used in simulator mode.
- Before you can erase flash memory of the target system, you must have written the data write and erase programs to the target system memory and executed the **c17 fls** command. If you execute the **c17 fle** command with no erase programs set, an error is assumed.
- Before writing data to flash memory of the target system, always be sure to use this command to erase flash memory.

## 10.7.13 Trace Command

### c17 tm (set trace mode)

[SIM]

#### Operation

Sets the conditions below.

#### Turning trace on/off

When you turn trace on, trace information is sampled along with program execution.

#### Trace information items to be displayed

You can choose the items in the trace information to be displayed.

#### Output destination of trace information

You can choose a window or file as the destination at which sampled trace information is output. Choosing a window displays trace information in the [Trace] window. Choosing a file requires that you specify a file name.

#### Format

**c17 tm on *Mode* [*Filename*]** (sets trace mode.)

**c17 tm off** (clears trace mode.)

*Mode:* Trace mode (contents of trace information displayed)

Specify within the range from 0x00 to 0xff. Set the bit corresponding to the item to be displayed to 1.

Bit 0 Trace number

Bit 1 Clock number

Bit 2 PC value and instruction code

Bit 3 Bus information (address, R/W and access size, data)

Bit 4 Register values (R0–R7, SP)

Bit 5 PSR value (IE, IL, CVZN)

Bit 6 Disassembled contents and source code

Bit 7 Cumulative number of clocks (number of clocks spent by each instruction when set to 0)

*Filename:* Name of file to which trace information is output

When a file name is specified, sampled trace information is output to the specified file, and not displayed in a window. When this entry is omitted, trace information is displayed in the [Trace] window, and not output to a file.

#### Usage example

##### ■ Example 1

```
(gdb) c17 tm on 0xff trace.log
```

This example sets the trace mode for displaying all information and specifies the `trace.log` file in which to save the information. Running the program after setting these options outputs trace information to a file for each instruction executed. If a file name is omitted, the information is displayed in the [Trace] window.

##### ■ Example 2

```
(gdb) c17 tm off
```

Trace mode is turned off. From this time on, no trace information is sampled even when running the program.

## Trace information

Running the target program after setting trace mode with this command displays trace information in the [Trace] window for each instruction executed, or outputs it to a file.

The contents of trace information displayed in a window or output to a file are as follows:

## Format of each trace information line

*num clk pc code bus\_addr/type/data r0 r1 r2 r3 r4 r5 r6 r7 sp ie/il/cvzn src\_mix*

**num:** Number of executed instructions (in decimal)  
 Number of instructions executed since the CPU was reset

**clk:** Number of execution clocks (in decimal)  
 Number of execution clocks since the CPU was reset

**pc:** Address of executed instructions (in hexadecimal)

**code:** Instruction codes (in hexadecimal)

**bus\_addr:** Accessed memory addresses (in hexadecimal)

**type:** Type of bus operation  
 r8: Byte data read; r16: 16-bit data read; r32: 32-bit data read  
 w8: Byte data write; w16: 16-bit data write; w32: 32-bit data write

**data:** Read/written data (in hexadecimal)

**r0–r7:** r0–r7 register values (in hexadecimal)

**sp:** sp register value (in hexadecimal)

**ie:** IE bit value in psr

**il:** IL bit value in psr

**cvzn:** C, V, Z and N bit values in psr

**src\_mix:** Disassembled contents and source codes of executed instructions

## Display example

## First half of information lines (trace number to register values)

num	clk	pc	code	bus	addr/type/data	r0	r1	r2	r3	r4	r5	r6	r7
652	1445	0040dc	9900	-----	---	000094	000000	000000	00ffff	000000	000000	000000	000000
653	1446	0040de	4000	-----	---	000094	000000	000000	00ffff	000000	000000	000000	000000
654	1447	0040e0	4000	-----	---	000094	000000	000000	00ffff	000000	000000	000000	000000
655	1449	0040e2	d900	000000	w16 00000000	000094	000000	000000	00ffff	000000	000000	000000	000000
656	1450	0040e4	2a12	-----	---	000094	000000	000000	00ffff	000000	000000	000000	000000
657	1451	0040e6	2814	-----	---	000000	000000	000000	00ffff	000000	000000	000000	000000
658	1452	0040e8	4000	-----	---	000000	000000	000000	00ffff	000000	000000	000000	000000
659	1457	0040ea	1805	003ef4	w32 000040ec	000000	000000	000000	00ffff	000000	000000	000000	000000
660	1458	0040f6	a001	-----	---	000000	000000	000000	00ffff	000000	000000	000000	000000
661	1459	0040f8	9000	-----	---	000000	000000	000000	00ffff	000000	000000	000000	000000
662	1462	0040fa	0e0e	-----	---	000000	000000	000000	00ffff	000000	000000	000000	000000
663	1466	004118	0120	003ef4	r32 000040ec	000000	000000	000000	00ffff	000000	000000	000000	000000
664	1467	0040ec	8201	-----	---	000000	000000	000000	00ffff	000001	000000	000000	000000
665	1468	0040ee	9205	-----	---	000000	000000	000000	00ffff	000001	000000	000000	000000

## Second half of information lines (SP value to source code)

sp	ie/il/cvzn	src	mix
003ef8	0 0 0010	ld	%r2, 0x0 (main.c) 00012 i = 0;
003ef8	0 0 0010	ext	0x0
003ef8	0 0 0010	ext	0x0
003ef8	0 0 0010	ld	[0x0], %r2
003ef8	0 0 0010	ld	%r4, %r2 (main.c) 00014 for( j = 0; j < 6; ++j )
003ef8	0 0 0010	ld	%r0, %r4 (main.c) 00016 sub(j);
003ef8	0 0 0010	ext	0x0
003ef4	0 0 0010	call	0x5
003ef4	0 0 0010	and	%r0, 0x1 (main.c) 00022 if(k & 0x1)
003ef4	0 0 0010	cmp	%r0, 0x0
003ef4	0 0 0010	jreq	0xe
003ef8	0 0 0010	ret	(main.c) 00027 }
003ef8	0 0 0000	add	%r4, 0x1 (main.c) 00014 for( j = 0; j < 6; ++j )
003ef8	0 0 1001	cmp	%r4, 0x5

## Notes

- This command cannot be used in ICD Mini mode.
- The number of clock cycles displayed in a window is calculated using the wait cycles information set in a parameter file. The information displayed may not be correct if the parameter file was set erroneously. For details, see Section 10.8, "Parameter Files".
- To change trace mode (with contents of trace information displayed), temporarily turn off trace mode (by executing `c17 tm off`), then set a new trace mode.

## 10.7.14 Simulated I/O Commands

### c17 stdin (data input simulation)

[ICD Mini / SIM]

#### Operation

Sets conditions for data to be entered from a file or [Simulated I/O] window and passed to the program.

The following conditions are set by the `c17 stdin` command:

- Break address (position at which **gdb** takes in data)
- Input buffer address (65-byte buffer)
- Input source (file or [Simulated I/O] window)

For operation on the program side, see Section 10.6.8, "Simulated I/O".

#### Format

```
c17 stdin 1 BreakAddr BufferAddr [Filename] (set)
c17 stdin 2 (clear)
```

*BreakAddr*: Break address (decimal, hexadecimal, or symbol)

*BufferAddr*: Input buffer address (decimal, hexadecimal, or symbol)

The buffer size is fixed to 65 bytes.

*Filename*: Name of input file

When this entry is omitted, data entry from the [Simulated I/O] window is assumed.

Conditions:  $0 \leq \text{BreakAddr} \leq 0\text{xfffff}$ ,  $0 \leq \text{BufferAddr} \leq 0\text{xfffff}$

#### Input examples

##### ■ Example 1

```
(gdb) c17 stdin 1 READ_FLASH READ_BUF input.txt
```

Data entered from a file is set.

When you run the program continuously after making this setting, the debugger aborts processing at the position of the `READ_FLASH` label in the program. Here, the debugger takes one line of data from the `input.txt` file and places it in the input buffer (`READ_BUF`), then resumes program execution.

##### ■ Example 2

```
(gdb) c17 stdin 1 READ_FLASH READ_BUF
```

Data entered via the [Simulated I/O] window is set.

When you run the program continuously after making this setting, the debugger aborts processing at the position of the `READ_FLASH` label in the program and opens the [Simulated I/O] window. When you enter data in the window and press the [Enter] key, the debugger takes data from the window and places it in the input buffer (`READ_BUF`), then resumes program execution.

##### ■ Example 3

```
(gdb) c17 stdin 2
```

The data input simulation function is cleared. If data entered from a file was set, the specified file is closed.

#### Notes

- The break addresses specified by the `c17 stdin` command cannot duplicate those of software PC breaks. Be sure to clear software PC breaks before executing the `c17 stdin` command. Break addresses overlapping those of hardware PC breakpoints are accepted.
- If the break address is specified with an odd value, the specified address is adjusted to the 16-bit boundary by assuming `LSB = 0`.

## c17 stdout (data output simulation)

[ICD Mini / SIM]

### Operation

Sets conditions for data to be output from a specified output buffer to a file or the [Simulated I/O] window.

The following conditions are set by the `c17 stdout` command:

- Break address (position at which **gdb** outputs data)
- Output buffer address (65-byte buffer)
- Output destination (file or [Simulated I/O] window)

For operation on the program side, see Section 10.6.8, "Simulated I/O".

### Format

`c17 stdout 1 BreakAddr BufferAddr [Filename]` (set)

`c17 stdout 2` (clear)

*BreakAddr*: Break address (decimal, hexadecimal, or symbol)

*BufferAddr*: Output buffer address (decimal, hexadecimal, or symbol)

The buffer size is fixed to 65 bytes.

*Filename*: Name of output file

When this entry is omitted, data is output to the [Simulated I/O] window.

Conditions:  $0 \leq \textit{BreakAddr} \leq 0\text{xfffff}$ ,  $0 \leq \textit{BufferAddr} \leq 0\text{xfffff}$

### Input examples

#### ■ Example 1

```
(gdb) c17 stdout 1 WRITE_FLASH WRITE_BUF output.txt
```

Data output to a file is set.

When you run the program continuously after making this setting, the debugger aborts processing at the position of the label `WRITE_FLASH` in the program. Here, the debugger outputs data from a specified buffer (`WRITE_BUF`) to a specified file, then resumes program execution.

#### ■ Example 2

```
(gdb) c17 stdout 1 WRITE_FLASH WRITE_BUF
```

Data output to the [Simulated I/O] window is set.

When you run the program continuously after making this setting, the debugger aborts processing at the position of the label `WRITE_FLASH` in the program. Here, the debugger opens the [Simulated I/O] window and displays it with the data contained in a specified buffer (`WRITE_BUF`), then resumes program execution.

#### ■ Example 3

```
(gdb) c17 stdout 2
```

The data output simulation function is cleared. If data output to a file was set, the specified file is closed.

### Notes

- The break addresses specified by the `c17 stdout` command cannot duplicate those of software PC breaks. Be sure to clear software PC breaks before executing the `c17 stdout` command. Break addresses overlapping those of hardware PC breakpoints are accepted.
- If the break address is specified with an odd value, the specified address is adjusted to the 16-bit boundary by assuming `LSB = 0`.

## 10.7.15 Flash Writer Commands

### **c17 fwe** (erase program/data)

[ICD Mini]

#### Operation

This is a dedicated command for the flash writer of the S5U1C17001H.  
It erases the data erase/write program or write data and address information loaded in the S5U1C17001H.

#### Format

**c17 fwe 0** (erases write data.)  
**c17 fwe 1** (erases data/erase write program.)

#### Usage example

##### ■ Example 1

```
(gdb) c17 fwe 0
```

The storage area for flash write data and the address information in the S5U1C17001H are erased.

##### ■ Example 2

```
(gdb) c17 fwe 1
```

The storage area for the flash erase/write program and the entry information in the S5U1C17001H are erased.

#### Notes

- This command cannot be used in simulator mode.
- This command can be used with the ICD Mini (S5U1C17001H). It cannot be executed normally with the ICD board (an error message will not be displayed).



**c17 fwlp** (load program)

[ICD Mini]

**Operation**

This is a dedicated command for the flash writer of the S5U1C17001H.  
It loads the data erase/write program from the host into the S5U1C17001H.

**Format**

**c17 fwlp** *Filename EraseEntryAddr WriteEntryAddr* [*Comment*]

*Filename*: Name of data erase/write program file (Motorola S3 format file)  
*EraseEntryAddr*: Erase routine entry address (RAM address on the CPU side, in decimal or hexadecimal)  
*WriteEntryAddr*: Write routine entry address (RAM address on the CPU side, in decimal or hexadecimal)  
*Comment*: Comments to identify data/address information (may be omitted)  
 Conditions:  $0 \leq \textit{EraseEntryAddr} \leq 0\text{xfffffe}$  (A0 = 0),  $0 \leq \textit{WriteEntryAddr} \leq 0\text{xfffffe}$  (A0 = 0),  
 $0 \leq \text{comment size} \leq 128$  bytes

**Usage example**

```
(gdb) c17 fwlp writer.sa 0x90 0xb4
```

The data erase/write program for the flash writer (prepared in file `writer.sa`) is loaded in the S5U1C17001H, and start addresses of the erase and write routines are set to 0x90 and 0xb4, respectively.

**Notes**

- This command cannot be used in simulator mode.
- This command can be used with the ICD Mini (S5U1C17001H). It cannot be executed normally with the ICD board (an error message will not be displayed).
- The data erase/write program is 8KB or less in size.

**c17 fwld** (load data)

[ICD Mini]

**Operation**

This is a dedicated command for the flash writer of the S5U1C17001H.  
It loads the data to be written to flash memory from the host into the S5U1C17001H.

**Format**

**c17 fwld** *Filename EraseStartBlock EraseEndBlock EraseParam* [*Comment*]

*Filename*: Name of data file (Motorola S3 format file)

*EraseStartBlock*: Block at which to start erasing (flash on the CPU side, in decimal or hexadecimal)

*EraseEndBlock*: Block at which to complete erasing (flash on the CPU side, in decimal or hexadecimal)

*EraseParam*: Erase parameter

A parameter passed to the erase routine in an external routine call

*Comment*: Comments to identify data/address information (may be omitted)

Conditions:  $0 \leq \text{EraseStartBlock} \leq 0\text{xffffffff}$ ,  $0 \leq \text{EraseEndBlock} \leq 0\text{xffffffff}$

If  $\text{EraseStartBlock} = \text{EraseEndBlock} = 0$ , all blocks in flash memory are assumed to be erased.

$0 \leq \text{comment size} \leq 128$  bytes

**Usage example**

```
(gdb) c17 fwld sample.sa 0 0 0x200000
```

The range of flash memory to be erased (all blocks in flash memory) is set, and the flash write data prepared in file `sample.sa` is loaded in the S5U1C17001H.

**Notes**

- This command cannot be used in simulator mode.
- This command can be used with the ICD Mini (S5U1C17001H). It cannot be executed normally with the ICD board (an error message will not be displayed).
- The program data is 8MB or less in size.

**c17 fwdc** (copy target memory)

[ICD Mini]

**Operation**

This is a dedicated command for the flash writer of the S5U1C17001H.

It loads the data stored in target board memory into the S5U1C17001H so that the data can be written to flash memory.

**Format**

**c17 fwdc** *SourceAddr* *Size* *EraseStartBlock* *EraseEndBlock* *EraseParam* [*Comment*]

*SourceAddr*: Target memory address from which to copy (flash on the CPU side, in decimal, hexadecimal, or symbol)

*Size*: Number of bytes to copy (decimal or hexadecimal)

*EraseStartBlock*: Block at which to start erasing (flash on the CPU side, in decimal or hexadecimal)

*EraseEndBlock*: Block at which to complete erasing (flash on the CPU side, in decimal or hexadecimal)

*EraseParam*: Erase parameter

A parameter passed to the erase routine in an external routine call

*Comment*: Comments to identify data/address information (may be omitted)

Conditions:  $0 \leq \textit{SourceAddr} \leq 0\text{xfffffe}$  ( $A0 = 0$ ),  $0 \leq \textit{Size} \leq 0\text{fffffe}$  ( $D0 = 0$ ),

$0 \leq \textit{EraseStartBlock} \leq 0\text{fffffff}$ ,  $0 \leq \textit{EraseEndBlock} \leq 0\text{fffffff}$

When  $\textit{EraseStartBlock} = \textit{EraseEndBlock} = 0$ , all blocks in flash memory are assumed to be erased.

$0 \leq \textit{comment size} \leq 128$  bytes

**Usage example**

```
(gdb) c17 fwdc FLASH_START 0x100000 0 0 0x200000
```

The range of flash memory to be erased (all blocks in flash memory) is set, and 1MB of area is copied from FLASH\_START in target memory to the S5U1C17001H.

**Notes**

- This command cannot be used in simulator mode.
- This command can be used with the ICD Mini (S5U1C17001H). It cannot be executed normally with the ICD board (an error message will not be displayed).
- The program data is 8MB or less in size.

**c17 fwd** (display flash writer information)

[ICD Mini]

**Operation**

This is a dedicated command for the flash writer of the S5U1C17001H.  
It displays information on the data and erase/write program loaded in the S5U1C17001H.

**Format****c17 fwd****Display**

```
(gdb) c17 fwd
CPU data address      : xxxxxxxx
Data size            : xxxxxxxx
Erase start block    : xxxxxxxx
Erase end block      : xxxxxxxx
Erase parameter      : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

CPU program address   : xxxxxxxx
Program size          : xxxxxxxx
Erase routine entry address : xxxxxxxx
Write routine entry address : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Information about the address and size of flash write data, and range of flash memory to be erased are displayed in the first half, as set by the `c17 fwd` or `c17 fwdc` command.

Information about the start address and size of the flash erase/write program, and entry addresses of the erase/write routines are displayed in the latter half, as set by the `c17 fwdp` command.

**Notes**

- This command cannot be used in simulator mode.
- This command can be used with the ICD Mini (S5U1C17001H). It cannot be executed normally with the ICD board (an error message will not be displayed).

## 10.7.16 Other Commands

### c17 log (logging)

[ICD Mini / SIM]

#### Operation

Saves the entered commands and command execution results displayed in the [Console] window to a file. The contents displayed in the [Console] window are written directly to a log file unchanged. Moreover, the contents of commands not displayed in the [Console] window, but executed from menus or by other means, are also output to a log file.

#### Format

```
c17 log Filename      (starts logging.)
c17 log                (completes logging.)
```

*Filename*: Name of log file

#### Usage example

##### ■ Example 1

```
(gdb) c17 log log.txt
log on
```

The commands to be entered hereafter and execution results are output to `log.txt` in text format. Log output remains on until the `c17 log` command is subsequently executed.

##### ■ Example 2

```
(gdb) c17 log
log off
```

The log file is closed and log output terminated.

#### Notes

- If an existing file name is specified, the `c17 log` command overwrites the file.
- To change the destination of log output to another file, terminate log output temporarily and specify a new file name before restarting log output.
- Any `c17 log` command contained in the command file selected by [Source...] on the [File] menu does not work. Load the command file using startup option `-x` of the debugger at startup or execute the `source` command in the [Console] window.
- The `c17 log` command contained in the command file specified by startup option `-x` is only effective with the [Console] window open. Be careful not to close the [Console] window before executing the `c17 log` command, because no log files will be created.

**source** (execute command file)

[ICD Mini / SIM]

**Operation**

Loads a command file and successively executes the debug commands written to the file.

**Format**

**source** *Filename*

*Filename*: Name of command file

**Usage example**

```
<File name = src.cmd>
# load symbol information
file /cygdrive/c/EPSON/gnu17/sample/tst/sample.elf
#connect to the debugger with specified mode and port
target sim
# load to memory
load /cygdrive/c/EPSON/gnu17/sample/tst/sample.elf
# reset
c17 rst
```

From # to the end of the line is interpreted as a comment.

```
(gdb) source src.cmd
(gdb)
(gdb) file /cygdrive/c/EPSON/gnu17/sample/tst/sample.elf
(gdb)
(gdb) target sim
boot () at boot.s:9
Connected to the simulator.
Current language: auto; currently asm
(gdb)
(gdb) load /cygdrive/c/EPSON/gnu17/sample/tst/sample.elf
Loading section .text, size 0xbc lma 0xc00000
Start address 0xc00000
Transfer rate: 1504 bits in <1 sec.
(gdb)
(gdb) c17 rst
CPU resetting ..... done
```

A specified command file is loaded and the commands contained in it are executed successively. The commands are displayed in the [Console] window as shown in the example above.

**GUI**

Choose [Source...] from the [File] menu in the [Source] window, select a command file in the ensuing dialog box, then click [Open]. The command file selected is loaded and the commands written to the file are executed.

**Notes**

- If the command file contains a description error, the debugger stops executing the command file there. Because no error messages appear in this case, be very careful when creating a command file.
- Once the `source` command is executed, it can be executed repeatedly by simply pressing the [Enter] key, as for other commands. In this case, all commands written to the command file are executed repeatedly.

**c17 clockmd** (set execution counter mode)**c17 clock** (display execution counter)

[ICD Mini / SIM]

**Operation**

**c17 clockmd**: Sets mode of the execution counter.

The counter can be set to cumulating mode (where measured values are cumulated until the counter is reset) or reset mode (where the counter is reset each time the program is run).

**c17 clock**: Displays the results counted during program execution.

In ICD Mini mode, the ICD execution counter value is displayed as hours, minutes, seconds, milliseconds and microseconds. The execution counter can measure up to about 6515 hours with a  $\pm 1$  microsecond of error. Note that it may not measure a short execution time (e.g. when instructions are executed for three microseconds or less).

In simulator mode, the counter value is displayed as the number of cycles.

For details about the execution counter, see "Measuring the execution cycles/execution time" in Section 10.6.5, "Executing the Program".

**Format**

**c17 clockmd Mode** (sets execution counter mode.)

**c17 clock** (displays execution counter.)

*Mode:* Counter mode

- 1 Reset mode
- 2 Cumulating mode (default)

**Usage example**

## ■ Example 1 (simulator mode)

```
(gdb) c17 clockmd 2
(gdb) c17 rst
CPU resetting ..... done
(gdb) continue
Continuing.

Breakpoint 1, sub (k=0) at main.c:20
(gdb) c17 clock
      218 cycle
(gdb) continue
Continuing.

Breakpoint 1, sub (k=1) at main.c:20
(gdb) c17 clock
      330 cycle
```

After setting in cumulating mode, the execution counter is reset by the reset command, thereby starting the program. In cumulating mode, the execution counter is not reset even when program execution is resumed after a break.

### ■ Example 2 (ICD Mini mode)

```
(gdb) c17 clockmd 1
(gdb) c17 rst
CPU resetting ..... done
(gdb) continue

Breakpoint 1, sub (k=0) at main.c:20
(gdb) c17 clock
      0 hour 0 min 1 sec 23 ms 1.33 us
(gdb) continue
Continuing.

Breakpoint 1, sub (k=1) at main.c:20
(gdb) c17 clock
      0 hour 1 min 53 sec 0 ms 0 us
```

In this example, after setting in reset mode, the execution counter is reset by the reset command, thereby starting the program. In reset mode, the execution counter is reset when program execution resumes after a break. Therefore, counts in this mode differ from those in cumulating mode.

#### Notes

- In ICD Mini mode, the `continue/until` command must be executed before this command becomes effective.  
In simulator mode, this command is always effective.
- The `c17 clock` command cannot display execution times while the lapse of time break (`c17 timebrk`) is enabled. Disable the lapse of time break (`c17 timebrk 0`) to measure execution times.
- The execution counter is reset in the following cases:
  1. When the `c17 clockmd` command changes execution counter mode (from cumulating mode to reset mode or vice versa)
  2. When the program is started with the counter set to reset mode
  3. When the CPU is reset
  4. When the `c17 timebrk`, `step`, `stepi`, `next`, `nexti` or `finish` command is executed in ICD Mini mode



**target** (connect target)

[ICD Mini / SIM]

**Operation**

Establishes connection to the target and sets connect mode.

ICD Mini mode: Connected with the ICD Mini (S5U1C17001H) or ICD board via a USB interface.

Simulator mode: Debugger is set to simulator mode.

**Format****target** *Type*

*Type*: One of the following symbols that specify the target

**icd usb**: Connected with the ICD via a USB interface (in ICD Mini mode).

**icd usb2**: Connected with the ICD via a USB interface (in ICD Mini mode).

This is used when debugging the target in an environment where a single PC and two ICD units are connected via USB. The debugger must be started up with the `--c17_double_starting` specified and use "target icd usb2" to connect the debugger. This allows you to start two debuggers with a single PC.

**sim**: Simulator started (in simulator mode).

**Usage example**

## ■ Example 1

```
(gdb) target sim
Connected to the simulator.
```

The debugger is set to simulator mode.

## ■ Example 2

```
(gdb) target icd usb
```

The debugger is set to ICD Mini mode.

**Notes**

When you set a memory map by loading a parameter file, be sure to execute the `c17 rpf` command before the `target` command. Also be sure to execute the `target` command before the `load` command, and the `file` command before the `target` command. The following shows the basic sequence of command execution:

```
(gdb) c17 rpf sample.par      (sets map information.)
(gdb) file sample.elf        (loads debugging information.)
(gdb) target icd usb         (this command)
(gdb) load                   (loads the program.)
(gdb) c17 rst                (resets the CPU.)
```

**detach** (disconnect target)

[ICD Mini / SIM]

**Operation**

Closes the port used to communicate with the target and exits the current connect mode.

**Format**

**detach**

**Usage example**

```
(gdb) target icd usb
      :
      Debug
      :
(gdb) detach
```

ICD Mini mode is exited.

**Notes**

This command can be used to turn the ICD off to switch between simulator mode and other modes, or perform operations on the target board. You need not execute this command to terminate debugging.

**pwd** (display current directory)

**cd** (change current directory)

[ICD Mini / SIM]

#### Operation

**pwd**: Displays the current directory.

**cd**: Changes the current directory.

#### Format

**pwd** (displays the current directory.)

**cd** *Directory* (changes the current directory.)

*Directory*: Character string used to specify a directory

#### Usage example

```
(gdb) pwd
Working directory /cygdrive/c/EPSON/gnu17/sample/tst.
(gdb) cd /cygdrive/c/EPSON/gnu17/sample/ansilib
Working directory /cygdrive/c/EPSON/gnu17/sample/ansilib.
```

After the current directory is confirmed, it is changed to "c:\EPSON\gnu17\sample\ansilib".

#### Notes

A drive name must be specified in "/cygdrive/*drive name*/" format. Do not specify a drive name in "c:" format. Moreover, use a slash (/) instead of (\) to delimit directories.

## c17 firmupdate (update firmware)

[ICD Mini]

### Operation

Updates the firmware written in the flash memory on the ICD. This command is effective after an ICD is connected. Therefore, the `target` command must be executed before this command can be used.

After the firmware has been updated, close `gdb` and turn the ICD off and on again.

### Format

`c17 firmupdate Filename`

*Filename*: Name of firmware file (Motorola S3 format file)

### Usage example

```
(gdb) target icd usb  
(gdb) c17 firmupdate icd17dmt.sa
```

Loads the firmware file `icd17dmt.sa` and updates the ICD firmware with the loaded contents.

### Notes

The `c17 firmupdate` command is effective only in ICD Mini mode.

## c17 ttbr (set TTBR)

[SIM]

### Operation

Sets an address to TTBR.

When the reset command (`c17 rst`) is executed, the value (reset vector) that has been stored in the address represented by TTBR is set to the PC. This command has the same function as the TTBR line written in the parameter file.

### Format

`c17 ttbr Address`

*Address*: Address to be set to TTBR (decimal, hexadecimal, or symbol)

Conditions:  $0 \leq \textit{Address} \leq 0\text{xffff}00$  (The eight low-order bits of *Address* must be 0x00.)

### Usage example

```
(gdb) c17 ttbr 0x8000
```

Sets address 0x8000 to TTBR.

### Notes

- The `c17 ttbr` command can be used only in simulator mode.
- This command must be executed before the `target` command.

**c17 help** (help)

[ICD Mini / SIM]

**Operation**

Displays a command description.

**Format**

**c17 help** [*Command*]

**c17 help** [*GroupNo.*]

*Command*: Name of command

*GroupNo.*: Command group number

**Usage example****■ Example 1**

```
(gdb) c17 help
group 0: memory ..... c17 fb,c17 fh,c17 fw,x /b,x /h,x /w,set {char},...
group 1: register ..... info reg,set $
group 2: execution ..... continue,until,step,stepi,next,nexti,finish,...
group 3: CPU reset ..... c17 rst
group 4: interrupt ..... c17 int,c17 intclear,c17 int_load
group 5: break ..... break,tbreak,hbreak,thbreak,delete,clear,enable,...
group 6: symbol ..... info locals,info var,print
group 7: file ..... file,load
group 8: map ..... c17 rpf,c17 map,c17 ttbr
group 9: flash memory ..... c17 fls,c17 fle
group 10: trace ..... c17 tm
group 11: simulated I/O .... c17 stdin,c17 stdout
group 12: flash writer ..... c17 fwe,c17 fwlp,c17 fwld,c17 fwdc,c17 fwd
group 13: others ..... c17 log,source,c17 clockmd,c17 clock,c17 firmupdate,...
Please type "c17 help 1" to show group 1 or type "c17 help c17 fb" to get usage ...
```

When you omit parameters, a list of command groups is displayed.

**■ Example 2**

```
(gdb) c17 help 2
group 2: execution
continue          Execute continuously
until             Execute continuously with temporary break
step             Single-step every line
stepi            Single-step every mnemonic
next             Single-step with skip every line
nexti            Single-step with skip every mnemonic
finish           Quit function
c17 callmd       Set user function call mode
c17 call         Call user function
Please type "c17 help continue" to get usage of command "continue".
```

When you specify a command group number, a list of commands belonging to that group is displayed.

**■ Example 3**

```
(gdb) c17 help step
step: Single-step, every line [ICD/SIM]

usage: step [Count]
      Count: Number of steps to execute (decimal or hexadecimal)
             One step is assumed if omitted.
      Conditions: 1-0x7fffffff

example:
(gdb) step
(gdb) step 10
```

When you specify a command, a detailed description of that command is displayed.

#### ■ Example 4

```
(gdb) c17 help c17 rst
c17 rst: Reset
```

[ICD/SIM]

```
usage: c17 rst
```

```
example:
```

```
(gdb)c17 rst
The CPU is reset.
```

To display a C17 command, specify the command name including "c17".

#### Notes

- Executing the `help` command (that comes standard with `gnu`) instead of the `c17 help` command displays help for the command classes and commands set in the `gnu` debuggers. This debugger does not support all of these command classes or commands. Note that device operation cannot be guaranteed for commands not described in this manual.
- A mode list (e.g. [ICD/SIM]) appears in the usage display (see Examples 3 and 4) indicating the modes in which the command is effective.

ICD: The command can be used in ICD Mini mode (when the ICD Mini (S5U1C17001H) or ICD board is used)

SIM: The command can be used in simulator mode (when debugging with the PC alone)

If "[ICD]" is displayed, it indicates that the command cannot be executed in modes other than ICD Mini mode.

**quit** (quit debugger)

[ICD Mini / SIM]

**Operation**

Terminates the debugger.  
Any ports or files used by the debugger that remain open are closed.

**Format**

`quit`  
`q` (*abbreviated form*)

**Usage example**

(gdb) `q`

**GUI**

You also can choose [Exit] from the [File] menu in the [Source] window or click the [Close] box in the [Source] window to quit the debugger.



## 10.8 Parameter Files

Parameter files are text files in which memory map information of the target system is written. The debugger reads this file to create memory map information, based on which it performs the following processing:

- Checks whether software PC break addresses are within a valid mapped area
- Breaks at write operation to ROM area (only in simulator mode)
- Breaks at accessing undefined areas (only in simulator mode)
- Breaks when stack overflows (only in simulator mode)
- Refers TTBR at reset.

When a parameter file is loaded, the appropriate size of storage (required for all areas of memory written to the file) is reserved in internal memory of your computer.

### How to load a parameter file

A parameter file is loaded in the debugger by executing the `c17 rpf` command.

```
(gdb) c17 rpf Filename.par (loads a parameter file to set a memory map.)
```

The **IDE** may be used to create a special command file, like the one loaded when the debugger starts. For details about **IDE**, see Chapter 5, "GNU17 IDE".

Be sure to execute the `c17 rpf` command before the `file`, `target`, and `load` commands. The following shows the basic sequence of command execution:

```
(gdb) c17 rpf sample.par (sets map information.)
(gdb) file sample.elf (loads debugging information.)
(gdb) target sim (connects the target.)
(gdb) load (loads the program.)
(gdb) c17 rst (resets the CPU.)
```

### How to create a parameter file

You can create parameter files by selecting [GNU17 Parameter Settings] from the [Properties] dialog box of the **IDE**. For details about **IDE**, see Chapter 5, "GNU17 IDE". Because parameter files are text files, you can use a general-purpose editor to create and correct parameter files.

**Note:** Do not use non-ASCII (Japanese, etc.) characters for file names (including extensions) and text in a file.

### Contents of parameter file

The following shows an example of a parameter file.

```
#gnu17 gdb parameter file (1)
ESSIM S1C17701 (2)
TTBR 8000 (3)

RAM 000000 001fff 00W #IRAM (4)
IO 040000 04ffff 00H #IO
RAM 600000 6ffffff 11H #RAM
ROM c00000 cffffff 55B B #ROM
STACK 000000 001fff #STACK (5)
```

#### (1) Comment

From # to the end of the line is interpreted as a comment.

#### (2) Target model

This parameter specifies the target model to simulate with the **ES-Sim17**.

#### (3) TTBR

This parameter specifies the trap table base address. The set value is referenced when the reset command (`c17 rst`) is executed and is used as the trap table (vector table) start address. The reset vector written in this address will be loaded to the PC for booting the system.

## (4) Memory map information

Each line is comprised as follows:

*Device StartAddr EndAddr [Condition] [BigEndian] [#Comment]*

*Device:* Specify the type of memory by using one of the following symbols:

**ROM** Write-only memory area  
**RAM** Readable/writable memory area  
**IO** Peripheral circuit control memory area

*StartAddr:* Specify the start address of the area.

The address must be specified in hexadecimal, but need not be preceded by 0x.

*EndAddr:* Specify the end address of the area.

The address must be specified in hexadecimal, but need not be preceded by 0x.

*Condition:* Specify wait cycles and a device size as shown below.

<Wait cycles for read><Wait cycles for write><Device size>

Wait cycles: **0** to **f** 0 to 15 cycles

Device size: **B** 8 bits  
**H** 16 bits  
**W** 32 bits

For example, assume a 16-bit device with one wait cycle for read and two wait cycles for write.

In this case, specify 12H.

This parameter is only effective in simulator mode, and may be omitted. When this entry is omitted, this parameter is processed as 77H.

*BigEndian:* For big endian devices, specify **B**.

This parameter is only effective in simulator mode. However, internal ROM, RAM, and I/O cannot be set to big endian format. When this entry is omitted, this parameter is processed as little endian. In ICD Mini mode, this parameter is ignored.

*#Comment:* A comment beginning with a # can be described in each line. However, required parameters must be described before writing a comment.

**Notes:** • Items *Device*, *StartAddr*, and *EndAddr* in memory map information cannot be omitted.

- If duplicate memory areas are specified, only the first area specified is effective.

RAM	600000	6ffffff	11H	#RAM	Effective
ROM	600000	6ffffff	22H	#ROM	Ignored
IO	600000	7ffffff	33H	#I/O	0x700000 through 0x7ffff effective as IO area

- The area size in the memory map should be specified with 256MB or less per area. Setting a larger size causes the debugger to fail the memory allocation during starting up.

## (5) Stack area information

Specify the area to be used as a stack in the format shown below.

*STACK StartAddr EndAddr*

*StartAddr:* Specify the start address of the area.

The address must be specified in hexadecimal, but need not be preceded by 0x.

*EndAddr:* Specify the end address of the area.

The address must be specified in hexadecimal, but need not be preceded by 0x.

This setting is effective in simulator mode, and causes a break to occur when the stack overflows.

In no case will this setting affect SP operation by a program.

## When not loading a parameter file

Parameter files are not always needed for debugging. You can perform debugging with any parameter file loaded in the debugger. In this case, however, the following limitations apply:

### In simulator mode

- If the `target sim` command is executed without executing the `c17 rpf` command, the simulation memory is reserved assuming that the RAM is mapped to the area from address 0x0 to address 0xfffff (16MB). In this case, the initial value of memory is 0x00. The TTBR address is set to 0x100000.
- The following map breaks do not work:
  1. Break by write operation to ROM area
  2. Access to an undefined area
  3. Stack overflow
- Counts of the execution counter and the number of clocks in trace information do not indicate the correct values.

## 10.9 Status and Error Messages

### 10.9.1 Status Messages

When the target program breaks, one of the following messages is displayed, indicating the cause of the break immediately before entering the command input wait state.

Table 10.9.1.1 Status messages

Message	Description
Breakpoint #, <i>function at file:line</i>	Made to break at a set breakpoint
Break by accessing no map.	Made to break by accessing unmapped area in simulator mode
Break by writing ROM area.	Made to break by accessing read-only area in simulator mode
Break by stack overflow.	Made to break by stack overflow in simulator mode
Illegal instruction.	Made to break by executing invalid instruction in simulator mode
Illegal delayed instruction.	Made to break by executing invalid delayed instruction in simulator mode
Break by key break.	Forcibly made to break by [Stop] button (in simulator mode)
Break by key break. Program received signal SIGINT, Interrupt.	Forcibly made to break by [Stop] button (in ICD Mini mode)

### 10.9.2 Error Messages

Table 10.9.2.1 Error messages (in alphabetical order)

Message	Description
... gdb : unrecognized option ' <i>option</i> '	An illegal startup option is specified.
A setup of a serial port was not completed.	An ICD mode not supported in <b>gdb</b> is specified.
Address is 24bit over.	The specified address is out of the 24-bit range. The maximum S1C17 address size is 24 bits (0xfffff).
Address(0x#) is ext or delayed instruction.	The specified address cannot be set due to an <code>ext</code> or delayed instruction.
C17 command error, command is not supported at present mode.	The input command cannot be executed in the current connect mode (ICD Mini or simulator mode).
C17 command error, command is too long.	The input command exceeds 256 characters in length.
C17 command error, invalid command.	The command is erroneous.
C17 command error, invalid parameter.	The command is specified with an invalid parameter.
C17 command error, number of parameter.	The number of command parameters is incorrect.
C17 command error, start address > end address.	The specified start address is greater than the end address.
C17 command error, start cycle < end cycle.	The specified start cycle is greater than the end cycle.
Cannot access memory at address #	Address # cannot be accessed.
Cannot allocate memory.	The necessary size of memory area as specified by a parameter could not be reserved.
Cannot clear hard pc break(0x#).	The specified hardware PC break address is invalid; no breakpoints are set there.
Cannot clear soft pc break(0x#).	The specified software PC break address is invalid; no breakpoints are set there.
Cannot display clock counter. Now Timer break mode is on. Please timer break mode off.	The execution counter value cannot be displayed when the lapse of time break is enabled. Disable the lapse of time break before execution times can be measured.
Cannot display clock counter. Time measurement should use continue or until command.	The execution counter value cannot be displayed if the <code>continue</code> or <code>until</code> command has not been executed.
Cannot measure clock timer.	The program execution time cannot be measured as it is too short.
Cannot open file( <i>file</i> ).	Cannot open the file.
Cannot open ICD17 usb driver.	Failed to open the USB drive.
Cannot set hard pc break any more.	The number of hardware PC breakpoints set exceeds the limit (one location only).
Cannot set soft pc break any more.	The number of software PC breakpoints set exceeds the limit (up to 200).

Message	Description
Cannot set timer. Timer Conditions: 1<=Timer<=300000	Cannot set a lapse of time break as the specified time exceeds the valid range.
Cannot write file.	Cannot write to the file.
Clock timer overflow.	The counter overflows during clock measurement.
Communication error(bcc).	A BCC error occurred in the message received from ICD.
Communication system error(#).	Connection was severed while communicating with ICD.
Copy end address max(0x#) overflow.	The end address of the source to be copied exceeds the upper limit (0xfffff).
Copy start address max(0x#) overflow.	The start address of the source to be copied exceeds the upper limit (0xfffff).
CPU is running.	Cannot accept a command while the CPU is running.
Erase entry address max(0x#) overflow.	The flash erase routine address exceeds the upper limit (0xfffff).
Flash memory end address max(0x#) overflow.	The flash memory end address exceeds the upper limit (0xfffff).
Flash memory start address max(0x#) overflow.	The flash memory start address exceeds the upper limit (0xfffff).
ICD17 is busy(#).	ICD is in a busy state.
Initialization error of ICD17.	Failed to initialize the target.
Invalid ID error(0x#).	The <b>gdb</b> has transmit an invalid ID number. (Internal error)
Invalid parameter file(#.file).	The parameter file contains an error.
Invalid parameter file, start address > end address(#.file).	The start and end addresses set in the parameter file are invalid because the former is greater than the latter.
It has not connected with a target.	The ICD and target cannot be connected.
It is not c17 architecture ELF file.	The file specified with the <code>file</code> command is not an elf format file supported in S5U1C17001C.
Load end address max(0x#) overflow.	The end address of the program to be written to flash memory exceeds the upper limit (0xfffff).
Load max address(0x#) overflow.	The address to be loaded exceeds the maximum value.
Load motorola file format error.(file)	The specified Motorola file contains a format error.
Load size limit(0x#) overflow.	The size of the specified file exceeds the upper limit. <ul style="list-style-type: none"> <li>• Flash write/erase program      8K bytes - 1 byte (0x1fff)</li> <li>• Write data for flash memory    3M bytes - 1 byte (0x2ffff)</li> <li>• Firmware                            8M bytes - 1 byte (0x7ffff)</li> </ul>
Load start address max(0x#) overflow.	The start address of the program to be written to flash memory exceeds the upper limit (0xfffff).
Receiving message is inaccurate.	A message exceeding the maximum size was received during communication with ICD.
Specification is required in the device for connecting.	The device name for ICD selection in the <code>target</code> command must be specified correctly.
Target down.	A communication error has occurred between the ICD and the target.
There is no argument given to this command.	Failed to disconnect the target.
Timeout error #(ICD17 -> host).	Wait for reception from ICD timed-out during communication with ICD.
Too much event(#).	The number of events specified with the <code>c17 int_load</code> (event file read) command exceeds the upper limit (256).
Transmitting failure(#).	NAK was received from ICD during communication with ICD.
USB communication error(host->ICD17).	Failed in USB transmission to ICD.
USB communication error(ICD17->host).	Failed in USB receiving from ICD.
Write entry address max(0x#) overflow.	The address of the flash write routine exceeds the upper limit (0xfffff).

ICD denotes the ICD Mini (S5U1C17001H) or ICD board.

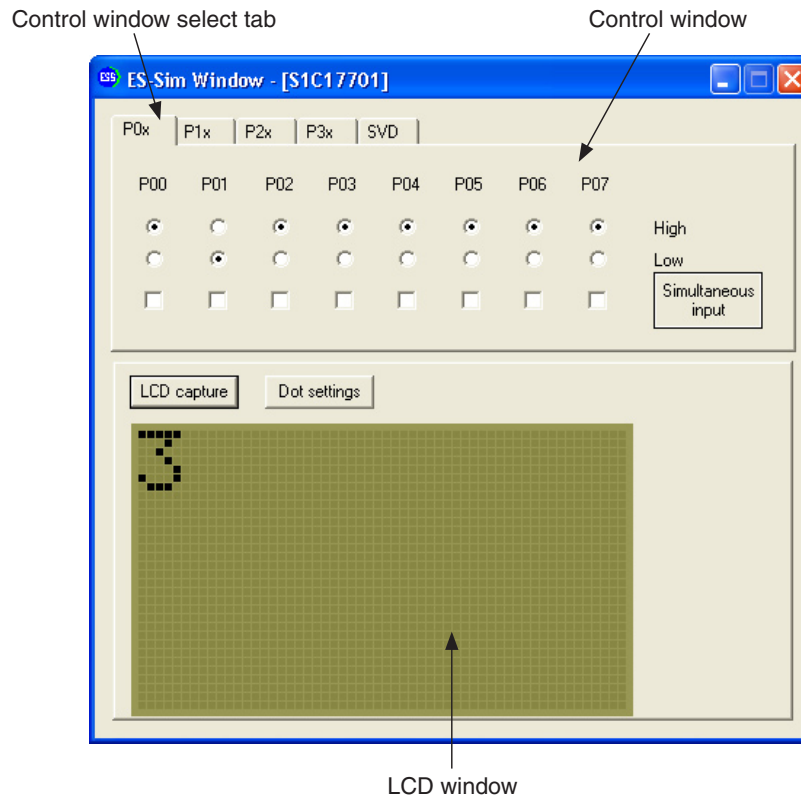
## 10.10 Embedded System Simulator (ES-Sim17)

The embedded system simulator (**ES-Sim17**) provides a feature to simulate the S1C17 hardware in a PC. It runs with simulator mode in the debugger **gdb** allowing practical debugging for application systems using a PC only.

The features of the **ES-Sim17** are as follows:

1. Indicates general-purpose port outputs status and simulates general-purpose port inputs.
2. Sets supply voltage level to evaluate the SVD operation.
3. Simulates LCD panel display by the LCD driver built into a target model.

The [ES-Sim] window shown below is used for all operations and display.



[ES-Sim] window (sample for S1C17701)

The **ES-Sim17** can simulate operations with the OSC1 clock in real time. For operations with the OSC3 clock, refer to "simulator\_readme.txt".

**Note:** The **ES-Sim17** is a simulator that runs on a PC, therefore, it has some restrictions. Refer to Section 10.10.7, "Restrictions", and "simulator\_readme.txt".

## 10.10.1 Input/Output Files

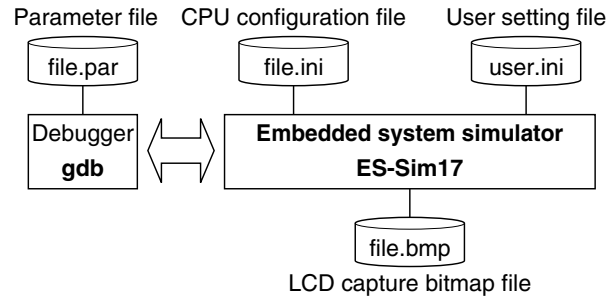


Figure 10.10.1.1 Input/output files

### Input files

#### Parameter file

File format: Text file

File name: `<file name>.par`

Description: This file has recorded in it the contents needed to set the memory map information for the debugger. (See Section 10.8, "Parameter Files".

The **ES-Sim17** obtains the target model name to be simulated from the parameter file that has been read in the debugger.

#### CPU configuration file

File format: Text file

File name: `<model name>.ini`

Description: This file contains the hardware configuration for the target model to be simulated in the **ES-Sim17**.

**Note:** Do not modify this file, as the **ES-Sim17** may not run normally.

#### User setting file

File format: Text file

File name: `user.ini` (fixed)

Description: This file contains the values that can be configured by the user.

Write the OSC1 and OSC3 clock frequencies in Hz units as below.

Example: <code>;;UserSetting</code>	The line beginning with ';' is regarded as a comment.
<code>;oscillator clock [Hz]</code>	
<code>[osc]</code>	OSC clock setting field
<code>osc1=32768</code>	OSC1 clock frequency = 32.768 kHz
<code>osc3=4000000</code>	OSC3 clock frequency = 4 MHz

### Output file

#### LCD screen-capture bitmap file

File format: Bitmap file

File name: `<file name>.bmp`

Description: This is a bitmap file that contains an LCD screen image simulated and can be generated by the **ES-Sim17**.

## 10.10.2 Starting and Terminating ES-Sim17

### Starting up ES-Sim17

The debugger launches the **ES-Sim17** when the following two conditions are met:

1. The parameter file read in the debugger has the comment below.

```
ESSIM <model name>
```

Example: ESSIM S1C17701

When the parameter file is created by the **IDE**, this comment is written to it according to the target processor selected.

2. The `target sim` command (to set the debugger in simulator mode) is executed.

When the **ES-Sim17** starts up, the [ES-Sim] window appears.

### Terminating ES-Sim17

The **ES-Sim17** terminates in the following two cases:

1. When the `detach` command is executed in the debugger
2. When the debugger is terminated

### Opening/closing the [ES-Sim] window

The [ES-Sim] window can be closed by clicking the [Close] button. (This operation does not terminate the **ES-Sim17**.)

To reopen the window, execute `\essim17\EssWnd.exe`. Note, however, that the **ES-Sim17** must be running at that point. If the **ES-Sim17** has already terminated, execute the `target sim` command again.

The [ES-Sim] window cannot be opened twice. If you attempt to open the window when it is already opened, the [ES-Sim] window moves to the foreground but a new window does not appear.

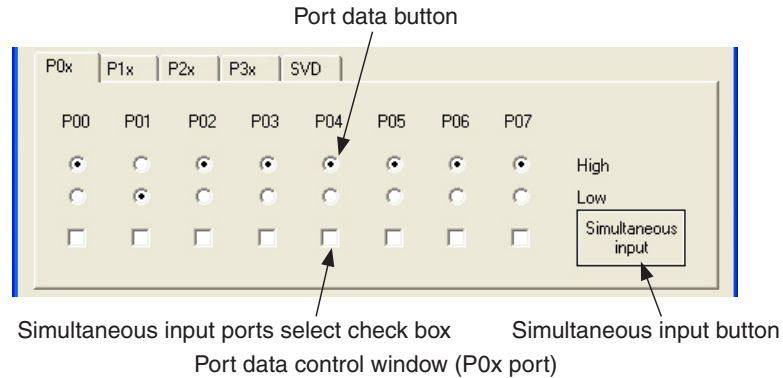


### 10.10.3 Simulating I/O Ports

The [ES-Sim] window allows control of the input status for the ports that have been set for general-purpose input. It also provides indicators to monitor the output status for the ports that have been set for general-purpose output.

#### Port data control window

Click on a control window select tab to select the port group (P0x, P1x, P2x, P3x) you want to operate or display.



The **ES-Sim17** obtains the information, such as selected I/O port functions and I/O directions, from the emulation memory in the PC to determine the port configuration to be displayed in the port data control window.

The port data buttons and simultaneous input ports select check boxes for the ports configured as general-purpose input becomes effective and are used to set input levels. When you change the input level in the window, the **ES-Sim17** updates the input data register in the emulation memory through the debugger.

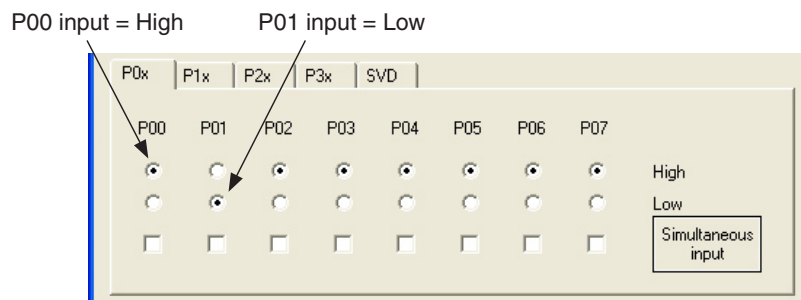
The port data buttons and simultaneous input ports select check boxes for the ports configured as a general-purpose output are grayed out to disable operations. However, the port data buttons indicate the current output status. When the output data register for the port configured as a general-purpose output is altered by the program, its status is reflected to the port data button.

The port data buttons and simultaneous input ports select check boxes for the ports configured to an internal peripheral input/output are not displayed.

The port data buttons and simultaneous input ports select check boxes for the ports that do not exist in the target model are not displayed.

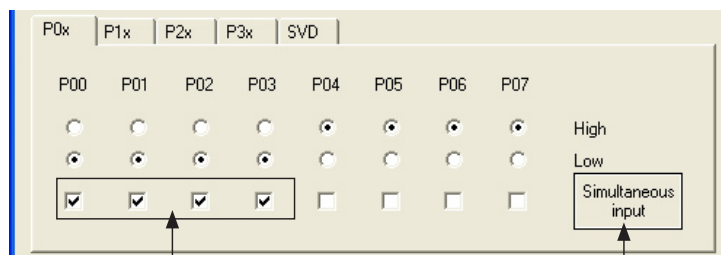
#### Setting the port input status

Select either High or Low port data button. This determines the current port input level.



## Simultaneous multiple key inputs

To simulate an operation press two or more keys simultaneously, first select the simultaneous input ports select check boxes for those ports. Then click the simultaneous input button. The port input levels are reversed from the status set with the port data buttons.



(1) Select the ports used for simultaneous input.

(2) Click the button to reverse the input levels.

This operation affects ports not contained in the tab page being currently displayed. The simultaneous input button located in any page reverses all the ports that have been selected with the simultaneous input ports select check boxes regardless of whether its tab page is displayed or not.

Even if multiple ports are selected with the simultaneous input ports select check boxes, the port data button can be used to control each port individually.

## Port output status

When a port changes its output level by executing the program in the debugger, the output status is reflected to the display of the port data button immediately.

The port data button for the ports configured as a general-purpose output cannot be operated using the mouse.

## P0 port key entry reset

If the target model supports the P0 port key entry reset function, the CPU can be reset by entering the active level signals to the ports specified with software. To evaluate this function, use the same way as the simultaneous multiple key inputs described above.

## Port input interrupts

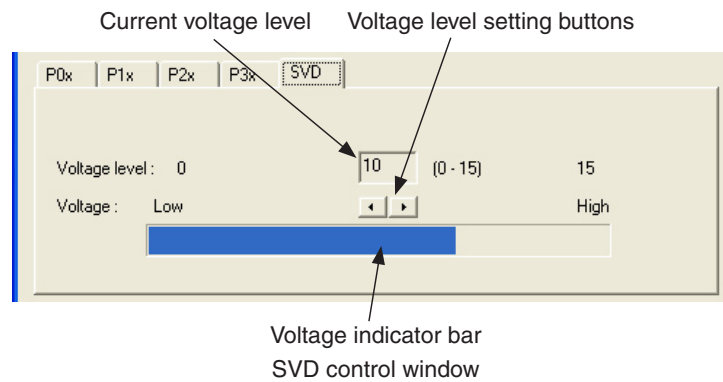
Changing the input status by an operation in the port data control window can generate a port input interrupt.

## 10.10.4 Simulating SVD

The [ES-Sim] window allows control of the supply voltage level for evaluating the SVD operation.

### SVD control window

Click on the SVD control window select tab to display the SVD control window.



The SVD control window is initialized with voltage level 15 (maximum level).

### Setting voltage level

The voltage level can be set within 16 steps\* from 0 (low) to 15 (high) using the voltage level setting buttons.

\* The number of voltage levels is equivalent to the number of valid SVD compare voltages supported in the target model. The number of available levels may be changed depending on the model.

Clicking the button changes the current voltage level and voltage indicator bar. At the same time, the compare voltage set in the SVD control register in the emulation memory and the voltage level set in this window are compared and the result is written to the SVD detection result register.

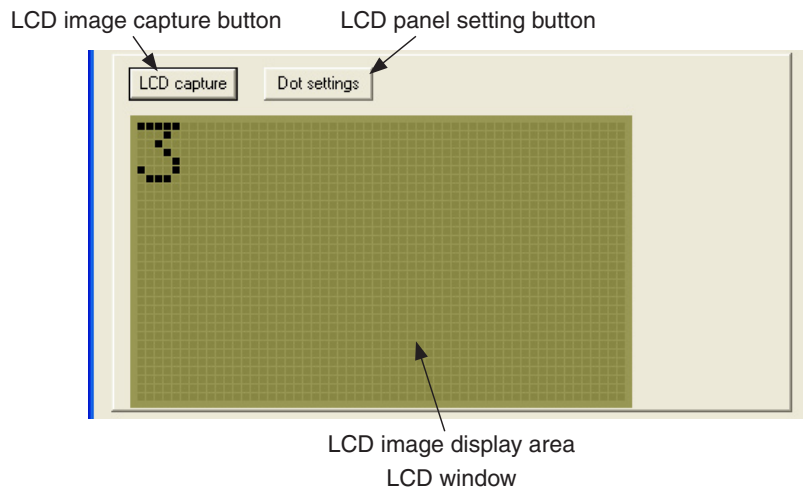
### SVD interrupt

If the target model supports the SVD interrupt, setting a voltage level lower than the SVD compare voltage in this window can generate an interrupt.

## 10.10.5 Simulating an LCD Panel

The **ES-Sim17** simulates display on an LCD panel according to control of the LCD driver and display memory.

### LCD window



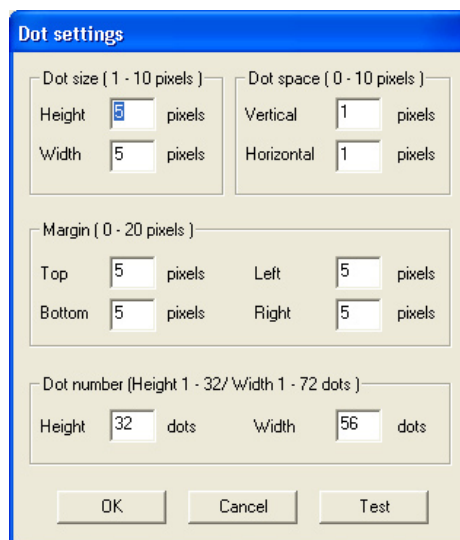
This window simulates display on a dot-matrix type LCD panel.

The **ES-Sim17** reads the contents of the display memory in 32-Hz cycles to redraw this window.

The drive duty setting and display control (display on/off, contrast adjustment, display area selection, etc.) in the program are reflected to this window.

### LCD panel configuration

The [Dot settings] dialog box displayed by the [Dot settings] button allows configuration of an LCD panel resolution, dot size to be displayed in the window and other conditions.



[Dot setting] dialog box

#### Dot size (1 - 10 pixels)

Set the dot size of the LCD panel to be displayed in the LCD window in PC display pixel units.

**Height** Set the dot height within the range from 1 to 10 pixels.

**Width** Set the dot width within the range from 1 to 10 pixels.

#### Dot space (0 - 10 pixels)

Set the clearance between dots in PC display pixel units.

**Vertical** Set the vertical spacing within the range from 0 to 10 pixels.

**Horizontal** Set the horizontal spacing within the range from 0 to 10 pixels.

**Margin (0 - 20 pixels)**

Set the top, bottom, left, and right margins in the LCD image display area in PC display pixel units.

Top	Set the top margin of the panel within the range from 0 to 20 pixels.
Bottom	Set the bottom margin of the panel within the range from 0 to 20 pixels.
Left	Set the left margin of the panel within the range from 0 to 20 pixels.
Right	Set the right margin of the panel within the range from 0 to 20 pixels.

**Dot number (Height 1 - 32/Width 1 - 72 dots)**

Set the LCD panel resolution.

Height	Set the number of vertical dots of the LCD panel within the range from 1 to 32 dots.
Width	Set the number of horizontal dots of the LCD panel within the range from 1 to 72 dots.

\* Up to the maximum number of dots supported by the LCD driver of the target model can be accepted.

[Test] button This button is used to check the LCD window configuration according to the settings above. The dialog box is left open and the changes are not actually applied to the window until the [OK] button is clicked.

[OK] button Clicking this button updates the LCD window according to the settings above and closes the dialog box.

[Cancel] button Clicking this button aborts the edit and closes the dialog box. The LCD window returns to the status previous to opening the dialog box if it has been changed with the [Test] button.

\* The [Test] and [OK] buttons are grayed out (disabled) if an invalid value is entered in the dialog box.

**Saving LCD screen**

The screen image being currently displayed in the LCD window can be saved to a bitmap file (.bmp).

Click the [LCD capture] button when the screen you want to capture is displayed. When the file save dialog box appears, select the directory and enter the file name you want to save.

The screen data is captured at the point the [LCD capture] button is clicked and the LCD window stops refreshing the display until the file save has completed.

The whole panel image is saved even if the LCD window does not display a part of the screen.

The **ES-Sim17** generates a Windows standard bitmap file (.bmp).

**Restrictions**

- The LCD window does not support segment type LCD panels.
- The dot size, contrast, and background color are different from those of the actual LCD panels.

## 10.10.6 ES-Sim17 Error Messages

Table 10.10.6.1 Error messages (displayed in the **gdb** [Console] window)

Message	Description
ES-Sim error 01 : Loading the dll file was failed.	The dll file for <b>ES-Sim17</b> does not exist in the default location or cannot be loaded normally.
ES-Sim error 02 : Opening the CPU construction file was failed.	The CPU configuration file does not exist in the specified location or cannot be loaded normally.
ES-Sim error 03 : Generating the CPU components was failed.	The <b>ES-Sim17</b> has failed generation of the CPU module as the CPU module definition is incorrect or no required dll file exists.
ES-Sim error 04 : Connecting the CPU components was failed.	The <b>ES-Sim17</b> has failed correction to the CPU module generation as the connect destination in the CPU module definition is incorrect.
ES-Sim error 05 : Opening the "user.ini" was failed.	The user setting file does not exist in the specified location or cannot be opened normally.
ES-Sim error 06 : Setting of the "user.ini" is invalid.	The setting value written in the user setting file is incorrect.

Table 10.10.6.2 Error messages (displayed in a dialog box)

Message	Description
Failed to save " <i>pathfile</i> ".	The <b>ES-Sim17</b> has failed saving the captured image to the file.

## 10.10.7 Restrictions

- The **ES-Sim17** supports the model shown below.  
S1C17701
  - The **ES-Sim17** supports monochrome dot-matrix LCD panels only as the external device.
  - The dot size, contrast, and panel color of the LCD window are different from those of the actual LCD panels.
  - The **ES-Sim17** performs simulation on an instruction cycle basis. Therefore, operation cycles lower than the instruction cycle cannot be simulated.
  - The **ES-Sim17** simulates the operation clock based on the instruction cycles. Therefore, the operation timings are not the same as those of the actual hardware.
  - The functions listed below cannot be simulated.
    1. Timer clock and oscillation clock external outputs
    2. Data transfer using the UART, I<sup>2</sup>C and SPI
    3. Noise and chattering filters
  - More than one **ES-Sim17** cannot be run on a PC for simulation.
  - Setting higher oscillation clock frequency causes degradation of simulation performance.
  - The I/O control registers that are not supported by the **ES-Sim17** function as general-purpose read/write registers. Also they are not initialized at a reset.
  - Some peripheral circuits, such as the oscillator and SVD circuits, need time until their operations stabilize. In the simulation by the **ES-Sim17**, they can operate with stability immediately after they start.
- \* For the restrictions in the latest version of **ES-Sim17** and model dependent restrictions, refer to "simulator\_readme.txt".

# 11 Other Tools

This chapter explains the other tools that are included in the S1C17 Family C Compiler Package.

## 11.1 make.exe

---

### 11.1.1 Functional Outline

The S1C17 Family C Compiler Package contains a make tool (hereafter referred to as the **make.exe**) that efficiently processes compilation to linkage.

Based on the dependence relationship between the sources written in a make file and the files output by each tool, the **make.exe** uses the necessary tools to update the files to the latest version. For example, if only one source file is corrected after the make process is completed once, the make executes compilation and/or assembly only for that file. For other modules, the make skips compilation and/or assembly processes for the source files and processes the object files from the linkage stage.

The **make.exe** in this package is based on gnu make (ver. 3.79), note, however, it only supports the dependency lists, suffix definitions, and macro definitions necessary to perform the above processing.

### 11.1.2 Input File

#### make file

File format: Text file

File name: *<file name>*.mak

Description: This file contains procedures for a make process. Normally use the make file created by the **IDE**.

### 11.1.3 Starting Method

#### General command line format

```
make [<option>] [<target name>]
```

The brackets [ ] denote that the specification can be omitted.

Example: `make -f test.mak clean`

#### Operation on IDE

This is called when a build is executed.

#### Option

The **make.exe** has the following available startup option.

**-f** *<file name>*

Function: **Specify make file**

Explanation: The **make.exe** reads in a make file specified by *<file name>* (extension included), and processes its contents.

Default: Unless the **-f** option is specified, a file named "makefile" is input as the make file.

#### Target name

Specify the target name (a label that indicates a command location) for the command to be executed. If this specification is omitted, the first target that appears in the make file is executed.

A make file created by the **IDE** contains a target name (`clean`) used to delete the files generated during make processing.

Example: `make -f test.mak clean`

When `clean` is specified, the object and map files that have been created by executing `test.mak` will be deleted. This function is useful to rebuild the program from all the source files.

To execute `clean` from the **IDE**, select [Clean] from the [Project] menu.

The S5U1C17001C supports only make processes without a target name specified or with the target name `clean` specified.



## 11.1.4 make Files

The make file is a text file that contains a description of the dependence relationship of the files and the commands to be executed.

Given below are examples of the make file generated by the IDE.

Example:

```
# Make file generated by Gnu17 Plug-in for Eclipse   Lines beginning with # are regarded as comments.
# This file should be placed directly under the project folder

# macro definitions for target file
TARGET= sample

# macro definitions for tools
TOOL_DIR= c:/EPSON/gnu17
CC= $(TOOL_DIR)/xgcc
CXX= $(TOOL_DIR)/xgcc
AS= $(TOOL_DIR)/xgcc
LD= $(TOOL_DIR)/ld
RM= $(TOOL_DIR)/rm
SED= $(TOOL_DIR)/sed
CC_KFILT= $(TOOL_DIR)/xgcc_filt

# macro definitions for tool flags
CFLAGS= -B$(TOOL_DIR)/ -O -gstabs -I$(TOOL_DIR)/include -fno-builtin -c
CXXFLAGS= ${CFLAGS}
ASFLAGS= -B$(TOOL_DIR)/ -c -xassembler-with-cpp -Wa,--gstabs
LDFLAGS= -Map sample.map -N -T sample_gnu17IDE.lds

# macro definitions for object files
OBJS= boot.o \
      main.o \

# macro definitions for library files
OBJLDS= $(TOOL_DIR)/lib/24bit/libstdio.a \
        $(TOOL_DIR)/lib/24bit/libc.a \
        $(TOOL_DIR)/lib/24bit/libgcc.a \
        $(TOOL_DIR)/lib/24bit/libc.a \

# macro definitions for dependency files
DEPS= $(OBJS:%.o=%.d)
SED_PTN= 's/[[[:space:]]\([a-zA-Z]\)\:\/ \\/cygdrive\/\1/g'

# macro definitions for creating dependency files
DEPCMD_CC= @$ (CC_KFILT) -M -MG $(CFLAGS) $< | $(SED) -e $(SED_PTN) >$(@:%.o=%.d)
DEPCMD_AS= @$ (AS) -M -MG $(ASFLAGS) $< | $(SED) -e $(SED_PTN) >$(@:%.o=%.d)

# targets and dependencies
.PHONY : all clean

all : $(TARGET).elf

$(TARGET).elf : $(OBJS) sample_gnu17IDE.mak sample_gnu17IDE.lds
    $(LD) $(LDFLAGS) -o $@ $(OBJS) $(OBJLDS)

## boot.s
boot.o : boot.s
    $(AS) $(ASFLAGS) -o $@ $<
    $(DEPCMD_AS)

## main.c
main.o : main.c
    $(CC_KFILT) $(CFLAGS) -o $@ $<
    $(DEPCMD_CC)

# include dependency files
-include $(DEPS)

# clean files
clean :
    $(RM) -f $(OBJS) $(TARGET).elf $(TARGET).map $(DEPS)
```

## Path descriptions in a make file

The make file supports descriptions in the cygwin format (or UNIX format). Therefore, the following precautions should be taken especially when a path is described.

### 1) Drive name and delimiter in path

"\" used in Windows must be replaced with "/". Additionally, write the drive name in the format "/cygdrive/<drive name>/".

Example:

```
c:/EPSON/gnu17/sample/tst/boot.o : c:/EPSON/gnu17/sample/tst/boot.s
as -o boot.o c:/EPSON/gnu17/sample/tst/boot.s
↓
/cygdrive/c/EPSON/gnu17/sample/tst/boot.o : /cygdrive/c/EPSON/gnu17/sample/tst/boot.s
as -o boot.o c:/EPSON/gnu17/sample/tst/boot.s
```

If a drive name is written in the form "<drive name>:", an error will result when make is executed. However, when entering a command line for Windows such as an assembler, you can use the "<drive name>:" format to write the path to be specified as a command line parameter.

### 2) Space

Make sure any spaces within a directory or file name is preceded by a "\".

Example: Tool Folder → Tool\ Folder

### 3) Case sensitive

Directory and file names are case sensitive. Be sure to check upper/lower case for path descriptions.

Also the **make.exe** allows descriptions of relative paths from the current directory in which the **make.exe** is invoked.

Example: .libraries/lib1.a

## Comments

A statement from # to the end of the line is regarded as a comment.

## 11.1.5 Macro Definition and Reference

You can define a character string as a macro in a make file and can refer to defined character strings using the macro names. The following shows the formats in which a macro can be defined and referenced.

Definition: **<macro name> = <character string>**

Reference: **\$( <macro name>)**

Example:

```
TARGET= sample
:
TOOL_DIR = /cygdrive/c/EPSON/gnu17
:
LD= $(TOOL_DIR)/ld
LIB_DIR= $(TOOL_DIR)/lib
:
LDFLAGS= -T $(TARGET).lds -Map $(TARGET).map -N
:
OBJS= boot.o \
      main.o \
:
OBJLDS=
:
LIBS= $(LIB_DIR)/libc.a $(LIB_DIR)/libgcc.a
:
$(LD) $(LDFLAGS) -o $@ $(OBJS) $(OBJLDS) $(LIBS)
```

The last line that refers macros in the example above is a command line to invoke the linker. If the macros have been defined as the example, this line will be executed after the macro names are replaced as below.

```
/cygdrive/c/EPSON/gnu17/ld -T sample.lds -Map sample.map -N -o sample.elf boot.o main.o
/cygdrive/c/EPSON/gnu17/lib/libc.a /cygdrive/c/EPSON/gnu17/lib/libgcc.a
```

### Predefined macros

`$@` used in the example above is a predefined macro. The following two predefined macros are available. Note, however, that they can be used only in the command lines in dependency lists.

**`$@`** This will be replaced with the target file name (including the extension) currently being processed.

Example:

```
sample.elf : . . .
    ld -o $@ . . .    (=ld -o sample.elf...)
```

**`$*`** This will be replaced with the target file name (not including the extension) currently being processed.

Example:

```
sample.elf : . . .
    ld $*.o . . .    (=ld sample.o...)
```

### Precaution

When the same macro name is defined twice or more, the newest defined macro is effective.

## 11.1.6 Dependency List

This section explains the dependency list when no suffix definition is used.

### Dependency list format

The make is executed according to a dependency list that is written in the following formats:

```
Format 1: <target file name>:<dependent file name 1> [ ^ <dependent file name2>... ]
      [      TAB          <command 1>
        TAB          <command 2>
                          :
                          ]
```

```
Format 2: <target name>:
      [      TAB          <command 1>
        TAB          <command 2>
                          :
                          ]
```

- ^ denotes a space.
- [ ] indicates that entries in brackets can be omitted.
- The command lines must begin with a TAB (space is not allowed).

#### Format 1

In Format 1, the dependent files necessary to obtain a target file are specified, and in cases when no target file has been created or there is a dependent file newer than the target file, the command that follows is executed.

Normally, a startup command of a tool is described as the command. The output file of the tool is specified as the target file and the input files are specified as the dependent files.

```
Example: main.o : $(SRC1_DIR)/main.c
           $(CC) $(CFLAGS) $(SRC1_DIR)/main.c
```

In this example, the target file `main.o` depends on `main.c`. If the target file `main.o` does not exist or `main.c` is newer than `main.o` (when the source is modified after it has been compiled), the command "`$(CC) $(CFLAGS) $(SRC1_DIR)/main.c`" (compilation by `xgcc`) is executed.

#### Format 2

If no dependent file is written, `<target name>` is used only as a label. By specifying a `<target name>` with the `make.exe` startup command, it is possible to execute the written command.

Example: Commands executed by `make -f test.mak clean`

```
clean:
    $(RM) -f $(OBJS) $(TARGET).elf $(TARGET).map $(DEPS)
```

If no `<target name>` is specified in the startup command, the first dependency list written in the file is used to execute the make process.

An executable command (with `.exe`) and its parameters can be written as a command. If no command has been written, nothing is executed. However, if a suffix definition with the extensions of the target file and the first dependent file is described, the command in the suffix definition is executed.

## Processing dependency lists by make.exe

For example, when `target.elf` is created from two source files, `boot.s` and `main.c`, the dependent relationship of the files including the temporary files (`.o`) is shown as Figure 11.1.6.1. Therefore, three dependency lists for `target.elf`, `boot.o` and `main.o` as the target files are required.

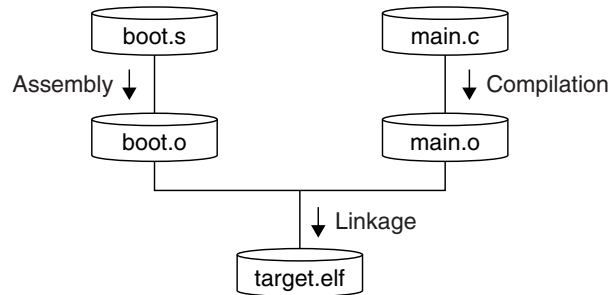


Figure 11.1.6.1 Relationship of files (example)

Sample of a make file:

```

target.elf : boot.o main.o
    $(LD) $(LDFLAGS) -o target.elf boot.o main.o $(LIBS) ] (A)

boot.o : boot.s
    $(AS) $(ASFLAGS) -o boot.o boot.s ] (B)

main.o : main.c
    $(CC) $(CFLAGS) main.c ] (C)
  
```

(A) Dependency list for generating `target.elf`

(B) Dependency list for generating `boot.o`

(C) Dependency list for generating `main.o`

\* See the above sample make file generated by the **IDE** for the macro contents referred with `$(XXX)`.

The first make process for this make file is executed as follows:

1. The **make.exe** checks Dependency list A (`target.elf: ...`) that appears first in the make file.
2. The dependent files `boot.o` and `main.o` are target files in other dependency lists, so the **make.exe** evaluates these dependency lists first.  
If `boot.o` or `main.o` does not exist and the dependency list for generating it is not written in the make file, an error occurs.
3. The **make.exe** evaluates Dependency list B (`boot.o: ...`). Then the command (for assembling `boot.s`) is executed to generate `boot.o` since `boot.o` does not exist at this time.  
If `boot.s` does not exist at this time, an error occurs since there is no dependency list for generating `boot.s`. In this case, create `boot.s` and locate it into the specified directory (the current directory in this make file) or delete the descriptions related to `boot.s` and `boot.o`.
4. Dependency list C (`main.o: ...`) is evaluated and `main.o` is generated similar to Step 3 above.
5. The make process returns to Dependency list A and the command (linkage) is executed since `target.elf` has not been generated yet. The `target.elf` is then generated.

If `main.c` is modified after the make process above has already completed, the next make is processed as follows:

1. The **make.exe** checks Dependency list A (`target.elf: ...`) that appears first in the make file.
2. The dependent files `boot.o` and `main.o` are target files in other dependency lists, so the **make.exe** evaluates these dependency lists first.
3. The **make.exe** evaluates Dependency list B (`boot.o: ...`). At this time, `boot.o` exists and it is newer than `boot.s`, so the following command (for assembling `boot.s`) is not executed.
4. The **make.exe** evaluates Dependency list C (`main.o: ...`). In this case, the dependent file `main.c` is newer than `main.o`, so the following command (for compiling `main.c`) is executed. This updates `main.o`.
5. The make process returns to Dependency list A and the command (linkage) is executed since the dependent file `main.o` is newer than `target.elf`. The `target.elf` is updated.

If no dependent file is updated from the previous make process, the commands in Dependency lists A to C are not executed.

### Precautions on writing dependency list

- In a dependency list, do not use the same file twice or more if possible. Otherwise, executing the command may set the time stamp of the file as a later time depending on the OS environment and the make sequence may not be processed normally.

Bad example:

```
vector2.o : vector.c
    /cygdrive/c/EPSON/gnu17/sed.exe -f ../comm/place.sed vector.c > vector2.c
    /cygdrive/c/EPSON/gnu17/xgcc -B/cygdrive/c/EPSON/gnu17/ -c vector2.c -o vector2.o
```

This example executes **sed.exe** to convert `vector.c` into `vector2.c` and then compiles `vector2.c` to generate `vector2.o`. It is better to separate into two dependency lists like below.

Good example:

```
vector2.o : vector2.c
    /cygdrive/c/EPSON/gnu17/xgcc -B/cygdrive/c/EPSON/gnu17/ -c vector2.c -o vector2.o
vector2.c : vector.c
    /cygdrive/c/EPSON/gnu17/sed.exe -f ../comm/place.sed vector.c > vector2.c
```

If modification is difficult, execute the **make.exe** again after the Make clean is executed.

- The relationship of dependency lists should be within 3 or 4 lists. Do not make a long link path of dependency lists.
- A maximum of about 4,000 dependency lists can be described in a make file. If descriptions exceed the limit, the make process may not be completed normally.
- Up to 255 alphanumeric characters can be used for a file name. 2-byte code characters are not allowed. Furthermore, when describing a file in full-path format, the file may not be accessed if the path exceeds 255 characters.

## 11.1.7 Suffix Definitions

Dependency lists in which the target file type is ".o" and the dependent file type is ".c" normally contain a command line to invoke the compiler. In other words, basically the same command line can be used common to all dependency lists that process the same file type if only the file names can be replaced. The suffix definition is a description of a list of file types (extensions) and commands to be executed and allows the make file to omit the description of commands in each dependency list. This function helps simplify dependency lists when many source files must be managed.

When a suffix definition has been made in the make file, the **make.exe** executes the commands described in the suffix definition for the dependency list that have the same file type configuration as the suffix definition and does not have a command description. If a dependency list has a command described, it is executed, not the command in the suffix definition. Therefore, the specific dependency list can execute a different command from others by describing the command in the normal form. This is useful when executing a different function only for the specific source, or when using a source located in a different directory from the other sources.

The following shows the dependency lists without a suffix definition and with a suffix definition.

### Dependency lists without a suffix definition

```
# dependency list start

### src definition start
SRC1_DIR= .
### src definition end

$(TARGET).elf : $(OBJS) $(TARGET).mak $(TARGET).lds
    $(LD) $(LDFLAGS) -o $@ $(OBJS) $(OBJLDS) $(LIBS)

## boot.s
boot.o : $(SRC1_DIR)/boot.s
    $(AS) $(ASFLAGS) -o boot.o $(SRC1_DIR)/boot.s

## main.c
main.o : $(SRC1_DIR)/main.c
    $(CC) $(CFLAGS) $(SRC1_DIR)/main.c

# dependency list end
```

### Dependency lists with a suffix definition

```
# suffix & rule definitions
.SUFFIXES : .c .s .o .elf

.c.o :
    $(CC) $(CFLAGS) -o $(SRC_DIR)/$*.o $(SRC_DIR)/$*.c
.s.o :
    $(AS) $(ASFLAGS) -o $(SRC_DIR)/$*.o $(SRC_DIR)/$*.s

# dependency list start

### src definition start
### src definition end

$(TARGET).elf : $(OBJS) $(TARGET).mak $(TARGET).lds
    $(LD) $(LDFLAGS) -o $@ $(OBJS) $(OBJLDS) $(LIBS)

## boot.s
boot.o : $(SRC_DIR)/boot.s

## main.c
main.o : $(SRC_DIR)/main.c

## sub.c
sub.o : $(SRC_DIR)/sub.c

# dependency list end
```

Suffix definition

Suffix rules

Dependency lists

## Format of a suffix definition

Specifying extensions

Format: **.SUFFIXES** : *.xxx .yyy .zzz .....*

Example: **.SUFFIXES** : *.c .s .o .elf*

Specify all the extensions related to the dependency lists to which the suffix rules are applied.

Definition of suffix rules

The following shows the format of a suffix rule:

Format: **.<extension of dependent file 1>.<extension of target file>** :

```

    TAB      <command 1>
[   TAB      <command 2>
      :
```

Example: **.c.o** :

```

    $(CC) $(CFLAGS) -o $(SRC_DIR)/$*.o $(SRC_DIR)/$*.c
```

- \$\* is a macro that will be replaced with the target file name (not including the extension) described in the dependency list.
- The command lines must begin with a TAB (space is not allowed).

The suffix rule in the example above corresponds to the dependency lists in the format below in which the target file type is ".o" and the type of the first dependent file is ".c".

*<file1>.o* : *<file1>.c* [*<other files>*]

The command in this suffix rule will be executed in the dependency list for which the command line is omitted.

Example: Dependency list

```

## main.c
main.o : $(SRC_DIR)/main.c

## sub.c
sub.o : $(SRC_DIR)/sub.c
```

The suffix rule (**.c.o**) in the example above is applied to these two dependency lists as follows:

```

## main.c
main.o : $(SRC_DIR)/main.c
    $(CC) $(CFLAGS) -o $(SRC_DIR)/main.o $(SRC_DIR)/main.c
## sub.c
sub.o : $(SRC_DIR)/sub.c
    $(CC) $(CFLAGS) -o $(SRC_DIR)/sub.o $(SRC_DIR)/sub.c
```

The time stamp of the dependent file is checked even when the suffix rule is applied and the command is not executed if the target file is newer than the dependent file.

## Precautions on use of suffix definition

When using a suffix definition, the target file name and the first dependent file name must be the same except for their extensions (also the file name is case sensitive).

Bad example: **main.o** : **main1.c**

In this case, the suffix rule is not applied. The make ignores such dependency lists and executes nothing for them.



### 11.1.8 clean

The make file created by **IDE** contains a description of a command to delete intermediate and object files other than the sources. This command can be executed by specifying the target name `clean` when the make is invoked (in **IDE**, select [Clean] from the [Project] menu.).

The following shows the command included in a make file:

Example:

```
TARGET= sample
:
TOOL_DIR = c:/EPSON/gnu17
:
RM= $(TOOL_DIR)/rm
:
OBJS= boot.o \
      main.o \
:
DEPS = $(OBJS:%.o=%.d)
clean:
  $(RM) -f $(OBJS) $(TARGET).elf $(TARGET).map $(DEPS)
```

"clean" uses the file remove utility (**rm.exe**) to delete the ".o" files, ".elf" file and ".map" file generated by a make process using this make file.

## 11.1.9 Messages

The following shows the messages generated by the **make.exe**:

Table 11.1.9.1 Normal message

Message	Description
make: ' <i>FILENAME</i> ' is up to date.	<i>FILENAME</i> has already been updated. The make process is terminated without executing a command.

Table 11.1.9.2 Error messages

Error message	Description
make: *** No rule to make target ' <i>FILENAME1</i> ', needed by ' <i>FILENAME2</i> '. Stop.	<i>FILENAME1</i> required for generating <i>FILENAME2</i> cannot be found, or a dependency list for generating <i>FILENAME1</i> has not been defined. The make process is terminated.
make: <i>TOOL</i> : Command not found	<i>TOOL</i> specified in the dependency list for generating <i>FILENAME</i> cannot be found. The make process is terminated.
make: *** [ <i>FILENAME</i> ] Error 127	
make: *** [ <i>FILENAME</i> ] Error 1	An error has occurred in the tool invoked. The make process is terminated.
make: Nothing to be done for ' <i>TARGET</i> '.	No process has been executed for creating the <i>TARGET</i> . This error will occur if no command is defined for creating a target file.
<i>XXX</i> .mak: <i>LINE</i> : *** missing separator. Stop.	There is an illegal separator symbol at the line <i>LINE</i> in the <i>XXX</i> .mak file. The make process is terminated. This error will occur if the command line in a dependency list or suffix rule begins with a character other than TAB, such as a space.

Table 11.1.9.3 Warning messages

Warning message	Description
make: *** Warning: File ' <i>FILENAME</i> ' has modification time in the future ( <i>yyyy-mm-dd hh:mm:ss</i> > <i>yyyy-mm-dd hh:mm:ss</i> )	The time stamp of <i>FILENAME</i> has been set at a later time. An erroneous clock setting has been made. The make process may not be completed normally.
make: warning: Clock skew detected. Your build may be incomplete.	

## 11.1.10 Precautions

- In the **make.exe** in this package, functions other than those described in this manual cannot be guaranteed to work normally.
- The make allows description of a maximum 30,000 characters for arguments of the executable files that can be invoked from a make file. Therefore, an error may occur during linkage if too many files that have long file names are added in the make file.

## 11.2 ccap.exe

---

### 11.2.1 Function

This tool produces a file from the messages output to the console (standard output or standard error) by other tools or commands.

### 11.2.2 Output File

#### Message file

File format: Text file

File name: *<file name>.err*

Description: This text file contains the tool messages saved through **ccap**.

### 11.2.3 Method for Using ccap

#### Startup format

```
ccap [<option>] <output file name> "<execution command>"
```

[ ] indicates the possibility to omit.

*<output file name>*: Specify a file name to which the messages to be output.

*<execution command>*: Input the startup command of the tool to be executed.

#### Options

The following four types of startup options are provided for **ccap**:

**-a**

Function: **Add to an existing file**

Explanation: If this option is specified, the output contents are added at the end of the specified file if it exists. If the file does not exist, **ccap** creates a new file.

Default: Unless this option is specified, the contents are overwritten to the specified file (if the file exists) or (if the file does not exist) **ccap** creates a new file.

**-o**

Function: **Output only a file**

Explanation: The messages of the executed command are output to a file only, and not output to the console.

Default: Unless this option is specified, the messages are output to both console and file.

**-c**

Function: **Disable outputting execution command line**

Explanation: If this option is specified, the execution command line is neither output to the console nor a file.

Default: Unless this option is specified, the execution command line is output along with messages.

**-e**

Function: **Error count**

Explanation: If this option is specified, **ccap** outputs the number of the error messages output by the executed command. The messages counted are those which begin with the following character strings:

Error      Count of the error messages

Warning    Count of the warning messages

Default: Error messages are not counted.

When entering an option, you need to place one or more spaces before and after the option.

Example: `c:\EPSON\gnu17\ccap -a -o -e Compile.err "xgcc -c -gstabs test.c"`

## Usage output

If no file name or execution command was specified or an option was not specified correctly, **ccap** ends after delivering the following message concerning the usage:

```
ccap
Console Capture Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
Usage:
  ccap [options] <output-file> "command line"
Options:
  -a : append mode
  -o : disable console output
  -c : disable command echo
  -e : display error count
Example:
  ccap -a -o -c console.cap "gcc sample.c"
```

### 11.2.4 Error Messages

The following shows the error messages generated by **ccap**:

Table 11.2.4.1 Error messages

Error message	Description
Error: Cannot execute	The specified "execution command" cannot be executed.
Error: Cannot open output file	The output file cannot be opened.

## 11.3 objdump.exe

---

### 11.3.1 Function

The **objdump** displays the internal data of binary files in elf format. Disassembled code, raw data, section configuration, section map addresses, data size and relocatable information symbol tables can be displayed.

### 11.3.2 Input Files

#### Executable object file

File format: Binary file in elf format

File name: *<file name>.elf*

Description: An executable object file after the linkage process by the linker has been completed. The contents will be displayed using the absolute addresses.

#### Object file

File format: Binary file in elf format

File name: *<file name>.o*

Description: An object file after assembled. The contents will be displayed using the relative addresses from the beginning of the file or section.

### 11.3.3 Method for Using objdump

#### Startup format

```
objdump <option> <input file name>
```

*<input file name>*: Specify a object file name to be dumped.

#### Options

The following startup options can be specified:

**-d**

Function: **Display disassembled contents**

Explanation: Displays all the executable sections after disassembling the object code. No source is displayed together.

**-h**

Function: **Display section information**

Explanation: Displays the section configuration, section size and address.

**-g**

Function: **Display information converted from debugging information**

Explanation: Displays the relations between sources and addresses based on the debugging information. The data types of the global symbols are also displayed.

**-t**

Function: **Display global symbol information**

Explanation: Displays a list of the global symbols including the local labels.

**-s**

Function: **Display in hexadecimal dump format**

Explanation: Displays all the section information in hexadecimal dump format. Data corresponding to unresolved symbols cannot be displayed correctly.

**-D**

Function: **Display disassembled contents for all sections**

Explanation: Displays all the sections after disassembling the object code.

## 11 OTHER TOOLS

**-G**

Function: **Display raw data of debugging information**

Explanation: Displays the raw data of the debugging information in stab format.

**-S**

Function: **Mixed display**

Explanation: Displays all the executable sections after disassembling the object code. The source code is also displayed with the corresponding disassembled code if possible.

When entering an option, you need to place one or more spaces before and after the option.

Example: c:\EPSON\gnu17\objdump -S test.elf

### 11.3.4 Dump Format

The following shows the display examples by specifying each option:

#### **-d (Disassembled display)**

Displays the disassembled information from the beginning of the executable section.

```
C:\EPSON\gnu17\>objdump -d main.o
main.o:      file format elf32-c17

Disassembly of section .text:

00000000 <main>:
   0: 3e5c ld.a      - [%sp],%r4      ld.a      - [%sp],%r4

00000002 <.LBB2>:
   2: 9900 ld        %r2,0x0          ld        %r2,0x0 <main>
   4: 4000 ext       0x0
   6: 4000 ext       0x0
   8: d900 ld        [0x0],%r2    xld       [0x0],%r2 <main>
   ....
```

#### **-h (Display section information)**

Displays the section configuration in the file, section size and mapped address (VMA and LMA). `.stab`, `.stabstr` and `.comment` are the sections that contains debugging information.

The load command does not send these debug information sections to the target.

```
C:\EPSON\gnu17\>objdump -h sample.elf

sample.elf:      file format elf32-c17

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .bss            000000b8  00000000  00000000  000000b4  2**2
    ALLOC
  1 .data           00000000  000000b8  00004080  00000134  2**0
    CONTENTS, ALLOC, LOAD, DATA
  2 .rodata         00000080  00004000  00004000  000000b4  2**2
    CONTENTS, ALLOC, LOAD, DATA
  3 .text           0000023e  00004080  00004080  00000134  2**1
    CONTENTS, ALLOC, LOAD, CODE
  4 .stab           00000bb8  000042c0  000042c0  00000374  2**2
    CONTENTS, READONLY, DEBUGGING
  5 .stabstr        000008b3  00004e78  00004e78  00000f2c  2**0
    CONTENTS, READONLY, DEBUGGING

*** Debugging sections will not be loaded to the target ***
```

**-g (Display converted debugging information)**

Displays the relations between the sources and the execution addresses in the following display format:

```
/* file <file name> line <source line number> addr 0x<address> */
```

The data types of the global symbols are also displayed.

```
C:\EPSON\gnu17\>objdump -g sample.elf

sample.elf:      file format elf32-c17

boot.s:
/* file boot.s line 47 addr 0x4080 */
/* file boot.s line 48 addr 0x4082 */
/* file boot.s line 49 addr 0x4084 */
/* file boot.s line 50 addr 0x4086 */
/* file boot.s line 53 addr 0x4088 */
.....
main.c:
typedef int16 int;
typedef int8 char;
typedef int32 long int;
typedef uint16 unsigned int;
typedef uint32 long unsigned int;
.....
typedef complex float0 complex float;
typedef complex float0 complex double;
typedef complex float0 complex long double;
typedef void *__builtin_va_list;
typedef enum { False, True } _Bool;
int main ()
{ /* 0x40a0 */
  /* file main.c line 10 addr 0x40a0 */
  { /* 0x40a2 */
    register int j /* 0x4 */;
    /* file main.c line 12 addr 0x40a2 */
    /* file main.c line 14 addr 0x40aa */
    /* file main.c line 16 addr 0x40ac */
    /* file main.c line 14 addr 0x40b2 */
  } /* 0x40b8 */
  /* file main.c line 18 addr 0x40b8 */
} /* 0x40bc */
.....
```

**-t (Display global symbol information)**

Displays the list of the global symbols including the internal symbols in the following display format:

SYMBOL TABLE:

```
<execution address> <local/global> <symbol type> <section> <data size> <symbol name>
```

```
C:\EPSON\gnu17\>objdump -t sample.elf
```

```
sample.elf:      file format elf32-c17

SYMBOL TABLE:
00000000 l d .bss      00000000
000000b8 l d .data      00000000
00004000 l d .rodata    00000000
00004080 l d .text      00000000
000042c0 l d .stab      00000000
00004e78 l d .stabstr   00000000
.....
```

**-s (Hexadecimal dump display)**

Displays the raw data of each section in the following display format:

```

Contents of section <section name>
<address> <raw data> <raw data> <raw data> <raw data> <ASCII characters>
.....
<address> <raw data> <raw data> <raw data> <raw data> <ASCII characters>
Contents of section <section name>
<address> <raw data> <raw data> <raw data> <raw data> <ASCII characters>
.....

```

```
C:\EPSON\gnu17\>objdump -s sample.elf
```

```

sample.elf:      file format elf32-c17

Contents of section .data:
Contents of section .rodata:
 4000 90400000 88400000 80400000 80400000  .@...@...@...@..
 4010 80400000 80400000 80400000 80400000  .@...@...@...@..
 4020 80400000 80400000 80400000 80400000  .@...@...@...@..
.....
Contents of section .text:
 4080 00000000 0000fc13 00000000 00002801  .....(
 4090 7e4000bc 0040b918 00400218 00402010  ~@...@...@...@ .
 40a0 5c3e0099 00400040 00d9122a 14280040  \>...@...*.(@
.....

```

**-D (Disassembled display for all sections)**

This option expands the display area by the `-d` option into all sections. The display format is the same as the `-d` option.

**-G (Display raw data of debugging information)**

Displays the raw data of the debugging information. Normally, this information is not used. Refer to the gnu documents or source codes for the display contents.

```
C:\EPSON\gnu17\>objdump -G sample.elf
```

```

sample.elf:      file format elf32-c17

Contents of .stab section:

Symnum n_type n_othr n_desc n_value  n_strx String

-1      HdrSym 0      249      000008b3 1
0       SO      0      0      00004080 1      boot.s
1       SLINE 0      47      00004080 0
2       SLINE 0      48      00004082 0
3       SLINE 0      49      00004084 0
4       SLINE 0      50      00004086 0
.....

```

**-S (Mixed source and disassembled code display)**

Displays the disassembled information of all the executable sections and the corresponding source codes together.

```
C:\EPSON\gnu17\>objdump -S sample.elf
```

```

sample.elf:      file format elf32-c17

Disassembly of section .text:

00004080 <__START_text>:
.text

```



```

.align 1

EXCEPTION:
    nop
    4080: 0000  nop
    .....
00004090 <BOOT>:

BOOT:
    xld.a  %sp, 0x3f00
    4090: 407e  ext    0x7e
    4092: bc00  ld.a   %sp,0x0      sld.a   %sp,0x3f00
    xcall  _init_lib
    4094: 4000  ext    0x0
    4096: 18b9  call   0xb9      xcall   0xb9      (0x00420A) <_init_lib>
    .....
000040a0 <main>:

void sub( int k );

main()
{
    40a0: 3e5c  ld.a   -[%sp],%r4      ld.a   -[%sp],%r4

000040a2 <.LBB2>:
    int j;
    i = 0;
    40a2: 9900  ld     %r2,0x0      ld     %r2,0x0 <__START_bss>
    40a4: 4000  ext   0x0
    40a6: 4000  ext   0x0
    40a8: d900  ld    [0x0],%r2    xld    [0x0],%r2 <__START_bss>
    .....
}
    40b8: 3e38  ld.a   %r4, [%sp]+  ld.a   %r4, [%sp]+
    40ba: 0120  ret
    .....

```

### 11.3.5 Error Message

The following shows the error message generated by **objdump**:

Table 11.3.5.1 Error message

Error message	Description
/cygdrive/X/path to objdump/objdump: filename: File format not recognized	An unrecognized file ( <i>filename</i> ) is specified. Specify an elf format file.

### 11.3.6 Precautions

- The disassembled display may be aborted halfway if the amount of information is too large.
- When a .o file before linking is dumped, the relative addresses from the beginning of each section are displayed, not the absolute addresses. In this case the beginning of each section is address 0x0.

## 11.4 objcopy.exe

---

### 11.4.1 Function

The **objcopy** is the gnu standard object file format conversion utility, and it copies and converts data format of object files.

In application development for the S1C17 Family, this tool is used to convert an elf format object file into Motorola S3 format HEX files so that data can be written to the ROMs.

Although **objcopy** supports many functions (options) and file formats, this section treats only the elf to HEX file conversion function. Refer to the documents for the gnu utilities for details of **objcopy**.

### 11.4.2 Input/Output Files

#### Input file

##### Object file

File format: Binary file in elf format

File name: *<file name>.elf*

Description: An executable object file after the linkage process by the linker has been completed.

#### Output file

##### HEX file

File format: Motorola S3 format file

File name: *<file name>.sa*

Description: A HEX data file for writing to the ROM. When the system uses two or more ROMs, create a data file for each ROM by extracting the section data to write to the ROM from the elf object file.

### 11.4.3 Method for Using objcopy

#### Startup format

```
objcopy <option> <input file name> [<output file name>]
```

[ ] indicates the possibility to omit.

<input file name>: Specify an elf format object file name to be converted.

<output file name>: Specify the Motorola S3 format HEX file name after conversion.

**Note:** When <output file name> is omitted, **objcopy** creates a temporary file used to output the converted data, and renames it with the input file name after the process has been completed. Therefore, the input file is destroyed.

#### Options

The following options are mainly used in application development for the S1C17 Family:

**-O srec**

Function: **Output in Motorola format**

Explanation: Specifies the Motorola format as the output file format.

**-O binary**

Function: **Output in binary format**

Explanation: Specifies the binary format as the output file format.

**--srec-forceS3**

Function: **Specify Motorola S3 format**

Explanation: Specifies the Motorola S3 format as the output file format. This option must be specified with the -O srec option.

Example: ... -O srec --srec-forceS3 ...

**-R SectionName**

Function: **Remove section**

Explanation: Specifies that the section named *SectionName* should not be included in the output file. This option can be specified multiple times in a command line.

**-v** (or **--verbose**)

Function: **Verbose output mode**

Explanation: Displays the converted object file names.

**-V** (or **--version**)

Function: **Display version number**

Explanation: Displays the version number of **objcopy**, and then terminates the process.

**--help**

Function: **Usage display**

Explanation: Displays the usage of **objcopy**, and then terminates the process.

### 11.4.4 Creating HEX Files

Open the command prompt window and execute **objcopy** at the command line as shown below.

```
c:\EPSON\gnu17\>objcopy -O srec -R SectionName --srec-forceS3 InputFile OutputFile
```

Running the above command converts sections other than those specified with the -R option into S3 records and generates an output file.

Example: Extract all section data from `input.elf` and write the data to `output.sa`.

```
c:\EPSON\gnu17\>objcopy -O srec --srec-forceS3 input.elf output.sa
```

## 11.5 ar.exe

---

### 11.5.1 Function

The **ar** is the gnu standard utility for maintenance of archived files. This utility is used to create and update library files that can be used with the linker **ld**. Refer to the documents for the gnu utilities for details of **ar**.

### 11.5.2 Input/Output Files

#### Object file

File format: Binary file in elf format

File name: *<file name>*.o

Description: A relocatable object file. The **ar** can add files in this format into an archive or extract an object from an archive to generate a file in this format.

#### Archive file (library file)

File format: Archive file in binary format

File name: *<file name>*.a

Description: A library file that can be input to the linker **ld**.

## 11.5.3 Method for Using ar

### Startup format

**ar** <key> [*<modifier>*] [*<add position>*] <archive> [*<objects>*]

[ ] indicates the possibility to omit.

<key>, <modifier>: Specify a process.

<add position>: Specify the location in the archive for inserting <objects> using the object name in the archive.

<archive>: Specify an archive file to be edited.

<objects>: Specify object file names to be added, extracted, moved or removed. Multiple file names can be specified by separating between the file names with a space.

### Keys

- d** Removes <objects> from the archive.
- m** Moves <objects> to the end of the archive. By specifying with modifier 'a' or 'b', the location in the archive where <objects> are moved can be specified.
- q** Adds <objects> at the end of the archive. This function does not update the symbol table in the archive.
- r** Replaces <objects> in the archive with the object files with the same name. If the archive does not contain <objects>, the <objects> files are added at the end of the archive. (By specifying with modifier 'a' or 'b', the location in the archive where <objects> are added can be specified.)
- t** Displays the list of objects in the archive or the list of the specified <objects>.
- x** Extracts <objects> from the archive and creates the object files. When <objects> are omitted, all the objects in the archive are extracted to create the files.

### Modifiers

- a** Use this modifier with key 'r' or 'm' to place <objects> behind the <add position>. Specify an object name located at the <add position> in the archive file.
- b** This modifier has the same function as 'a' but <objects> are placed in front of the <add position>.
- s** Forcibly updates the symbol table in the archive.
- u** Use this modifier with key 'r' to replace only the updated objects in the <objects> that are newer than those included in the archive.
- v** Specifies verbose mode to display the executed processes.

Do not enter a space between the keys and modifiers.

## Usage examples

### (1) Creating a new archive

```
ar rs mylib.a func1.o func3.o
(mylib.a: func1.o + func3.o)
```

When the specified archive (`mylib.a`) does not exist, a new archive is created and the specified object files (`func1.o` and `func3.o`) are added into it in the specified order.

### (2) Adding objects

```
ar rs mylib.a func4.o func5.o
(mylib.a: func1.o + func3.o + func4.o + func5.o)
func4.o and func5.o are added at the end of mylib.a.
```

### (3) Adding an object to the specified location

```
ar ras func1.o mylib.a func2.o
(mylib.a: func1.o + func2.o + func3.o + func4.o + func5.o)
func2.o is added behind the func1.o in mylib.a.
```

### (4) Replacing objects

```
ar rus mylib.a func1.o func2.o func3.o func4.o func5.o
(mylib.a: func1.o + func2.o + func3.o + func4.o + func5.o)
```

If there are files from among `func1.o`, `func2.o`, `func3.o`, `func4.o` and `func5.o` that have been updated after they have been added into `mylib.a`, the objects in `mylib.a` are replaced with the newer files. The objects that have not been updated are not replaced.

### (5) Extracting an object

```
ar x mylib.a func5.o
(mylib.a: func1.o + func2.o + func3.o + func4.o + func5.o)
func5.o is extracted from mylib.a and an object file is created. The archive is not modified.
```

### (6) Removing an object

```
ar ds mylib.a func5.o
(mylib.a: func1.o + func2.o + func3.o + func4.o)
func5.o is removed from mylib.a.
```

## 11.6 moto2ff.exe

---

### 11.6.1 Function

The **moto2ff** loads a Motorola S3 format file and fills the unused area in data of the file with 0xff to generate an output file. The start address and block size can be specified to output the required area only.

In an application development for the S1C17 Family, the **moto2ff** is used to retrieve ROM area data from the HEX file generated by the **objcopy**. The ROM area data generated by **moto2ff** should be processed with **sconv32** and **winmdc** to generate the mask data to be submitted to Seiko Epson. For the mask data generation procedure, refer to Section 3.3.7, "Creating ROM Data".

### 11.6.2 Input/Output Files

#### Input file

##### HEX file

File format: Motorola S3 format file

File name: *<file name>*.sa

Description: A Motorola S3 format HEX file converted from an elf format executable file by the **objcopy**.

#### Output file

##### ROM area data file

File format: Motorola S3 format file

File name: *<file name>*.saf

Description: A data file of the specified ROM area in which the unused area is filled with 0xff.

### 11.6.3 Startup Format

**moto2ff** *<data start address>* *<data block size>* *<input file name>*

*<data start address>*: Specify the data output start address in the input file using a hexadecimal number.

*<data block size>*: Specify the output data block size in bytes using a hexadecimal number.

*<input file name>*: Specify the file name of the Motorola S3 format file to be filled with 0xff.

The file name must be within 128 characters including a path and an extension. Path can be specified for the input file, note, however, that the output file will be located in the current directory.

- Usage will be displayed when no parameters are specified.
- If the output file already exists, it will be overwritten.
- When an error occurs, an error message is displayed and the output file is not generated.
- If the input Motorola S3 file contains data that exceeds the range specified by a start address and a block size, a warning message appears and the **moto2ff** generates the file with the data only within the specified area.
- If Motorola S3 data records are in the same address, the first data is overwritten by the last.
- Make sure that the data start address and data block size are correct values for the model by referring its technical manual. If an incorrect value is input, an error will occur in the **winmdc** process to generate final mask data.
- When **moto2ff** has completed successfully, the following message is shown in the standard output.  
Convert Completed

## 11.6.4 Error/Warning Messages

The following shows the error and warning messages generated by the **moto2ff**:

Table 11.6.4.1 Error messages

Error message	Description
Input filename is over 128 letters.	The input file name has exceeded 128 characters.
Cannot open input file " <i>FILENAME</i> ".	The input file <i>FILENAME</i> cannot be opened.
Cannot open output file " <i>FILENAME</i> ".	The output file <i>FILENAME</i> cannot be opened.
Motorola S3 checksum error.	A checksum error occurred while reading Motorola S3 format file.
Cannot allocate memory.	Cannot allocate memory.

Table 11.6.4.2 Warning messages

Warning message	Description
<i>FILENAME</i> contains data outside of specified range (" <i>STARTADDR</i> ":" <i>SIZE</i> ")	The input file <i>FILENAME</i> contains data that exceeds the specified range ( <i>SIZE</i> bytes from <i>STARTADDR</i> ). Data exceeded the specified range is not output.
Invalid file format in " <i>FILENAME</i> " line " <i>NUMBER</i> ".	The input file <i>FILENAME</i> contains an invalid format data at line <i>NUMBER</i> .

## 11.6.5 Creating ROM Area Data

After a Motorola S3 format HEX file has been generated by the **objcopy**, convert it into ROM area data using the **moto2ff**.

Open the command prompt window and execute **moto2ff** as shown below.

Example: C:\EPSON\gnu17\>moto2ff 8000 10000 input.sa

The command above outputs the data of 0x10000 bytes starting from address 0x8000 contained in *input.sa* to *input.saf*. The unused addresses within the range from addresses 0x8000 to 0x17fff are filled with 0xff.

The ROM area data file for the internal ROM are generated by the above procedure.

After that convert the ROM area data file generated here into the Motorola S2 format ROM data file using **sconv32**. Then perform the final verification of program operation on the actual target board using that file.

Finally, pack the verified ROM data file and the mask option data file generated by **winfo** into a mask data file using **winmdc** and present it to Seiko Epson.



## 11.7 sconv32.exe

### 11.7.1 Function

The **sconv32** is a tool to convert a Motorola S format into another S format. In an application development for the S1C17 Family, the **sconv32** is used to convert the Motorola S3 format ROM area data file generated by **moto2ff** into the Motorola S2 format.

The converted ROM data file should be processed with **winmdc** to generate the mask data to be submitted to Seiko Epson after performing the final verification of program operation on the actual target board using the ROM data file. For the mask data generation procedure, refer to Section 3.3.7, "Creating ROM Data".

### 11.7.2 Input/Output Files

#### Input file

##### ROM area data file

File format: Motorola S3 format file

File name: *<file name>.saf*

Description: A Motorola S3 format HEX file generated by **moto2ff**.

#### Output file

##### ROM data file

File format: Motorola S2 format file

File name: *<file name>.psa*

Description: The Motorola S2 format file converted from the input file.

### 11.7.3 Startup Format

**sconv32 S2 <input file name> <output file name>**

**S2:** This is a switch to convert the input file into the Motorola S2 format.

**<input file name>:** Specify a ROM area data file generated by **moto2ff**.

**<output file name>:** Specify the output file name that will be converted into the Motorola S2 format.

The file extension must be ".psa".

- Usage will be displayed when no parameters are specified.
- If the output file already exists, it will be overwritten.
- When an error occurs, an error message is output to the standard error device and the output file is not generated.
- The [ESC] key can be used to forcibly terminate the process while converting.
- When **sconv32** has completed successfully, the following message is displayed.

Converted Lines 130	Number of lines converted	*1
FILE CONVERT COMPLETE.	End message	*1
---	Input file information	*2
RECORD:---3---7--	S record type in the file	*2
COMPLEMENT:1	Checksum (one's-complement)	*2
---	Output file information	*2
RECORD:--2-----8-	S record type in the file	*2
COMPLEMENT:1	Checksum (one's-complement)	*2

\*1: Output to the standard error device

\*2: Output to the standard output device

## 11.7.4 Error Messages

The following shows the error messages generated by the **sconv32**:

Table 11.7.4.1 Error messages

Error message	Description
INVALID SWITCH.	An invalid switch is specified.
COMPLEMENT SWITCH ERROR.	The specified complement of the checksum for the output file is incorrect.
S FORMAT TYPE ERROR.	The specified S format for the output file is incorrect.
NO INPUT FILE NAME.	An input file is not specified.
NO OUTPUT FILE NAME.	An output file is not specified.
INPUT SAME FILE.	The same file name is specified for input and output.
CANNOT OPEN SOURCE FILE ( <i>filename</i> ).	The specified input file cannot be found or cannot be opened.
CANNOT OPEN DESTINATION FILE ( <i>filename</i> ).	The output file cannot be opened.
SOURCE RECORD TYPE NOT SUPPORT.	The input file has an unsupported record type.
ADDRESS LENGTH RANGE OVER.	The address range of the input file exceeds the address range for the S format to be converted.
OTHER ERROR.	Another error has occurred.

## 11.8 Outline of the Development Tools

---

The S1C17 C Compiler Package contains the tools to create mask option and mask data files, as well as files that contain descriptions of setup information for each type of microcomputer. The tools below are Windows GUI applications that run under Windows 2000 or Windows XP versions.

### 1. Function option generator <winfog.exe>

The **winfog** creates the function option document file that is necessary to generate IC mask patterns after selecting the mask options of the S1C17xxx. You can create function option data by selecting the appropriate item using the check boxes. Refer to Section 11.9 for details.

### 2. Mask data checker <winmdc.exe>

The **winmdc** checks the data in development-completed built-in ROM file and option document file to create the mask data file that will be presented to Seiko Epson. Refer to Section 11.10 for details.

### 3. Device information definition file <S1C17xxx.ini>

This file is used to set information, such as the configuration of options, on each type of microcomputer for the tools described above. This file must be available before each tool can be executed.

**Note:** There is no difference between each tool between the different types of microcomputers. Therefore, the explanations in this manual are for all types of microcomputers using "S1C17xxx" as the representative name. The contents of the sample screens shown in this manual vary according to the type of microcomputer.

For the mask data generation procedure including creation of a internal ROM data filer, refer to Section 3.3.7, "Creating ROM Data".

## 11.9 winfog.exe

### 11.9.1 Outline of winfog

The S1C17 Family allows several hardware specifications such as I/O port functions to be selected as mask options depending on the model. This helps you to configure the hardware of your product by changing the S1C17 chip's mask patterns according to its specifications.

The Function Option Generator **winfog** is the software tool for creating the files necessary to generate mask patterns. Its graphical user interface (GUI) ensures easy selection mask options. From the files created by **winfog**, Seiko Epson produces the mask patterns for the S1C17 chip.

### 11.9.2 Input/Output Files

Figure 11.9.2.1 shows the input/output files of **winfog**.

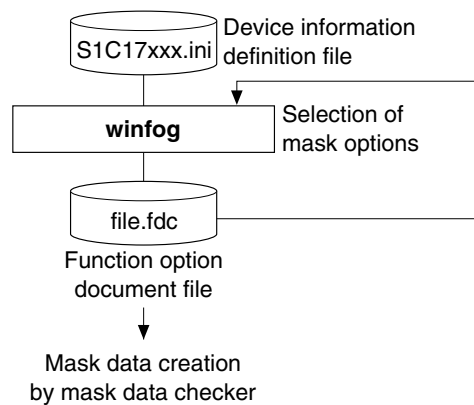


Figure 11.9.2.1 Input/output files of **winfog**

#### Input file

##### Device information definition file

File format: Text format file

File name: S1C17xxx.ini

Description: This file contains option lists for various types of microcomputers and other information. Always be sure to use the files presented by Seiko Epson. This file is effective for only the type of microcomputer indicated by the file name. Do not modify the contents of the file or use the file in other types of microcomputers.

#### Output file

##### Function option document file

File format: Text format file

File name: <file name>.fdc

Description: This is a text format file in which the contents of selected mask options are stored. You can read this file into **winfog** and correct the already selected option settings. This file is packed along with completed other program/data files into a single file by the mask data checker **winmdc**, which we would like to have presented to Seiko Epson as the mask data file. From this file, Seiko Epson will create the mask patterns for the IC.

## 11.9.3 Starting Up

### Startup from Explorer



Double-click on the **winfog.exe** icon or select **winfog** from the start menu.

If the device information definition file (S1C17xxx.ini) was loaded into your computer during previous execution, **winfog** automatically reads the same file as it starts.

Alternatively, drag the device information definition file icon into the **winfog.exe** icon to start **winfog**, which will then read the device information definition file.

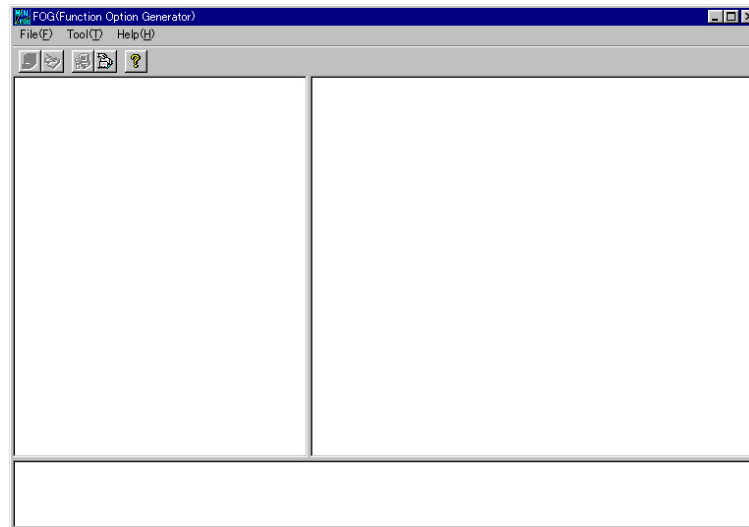
### Startup by command input

You can also start **winfog** from the command prompt by entering the command shown below.

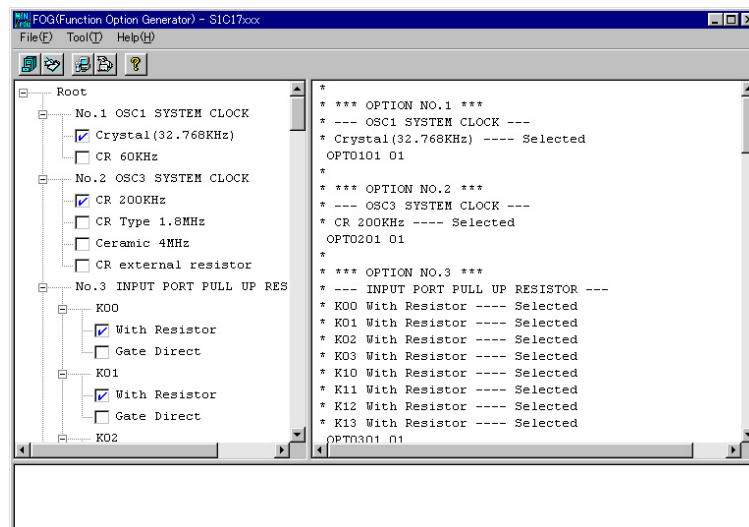
```
>winfog [s1c17xxx.ini]
```

You can specify the device information definition file (S1C17xxx.ini) as a command option. (You can also specify a path.) When you specify the device information definition file here, **winfog** reads it as it starts. This specification can be omitted.

When **winfog** starts, it displays the [FOG] window. The following diagrams show a [FOG] window when the device information definition file has been loaded and when it has not.

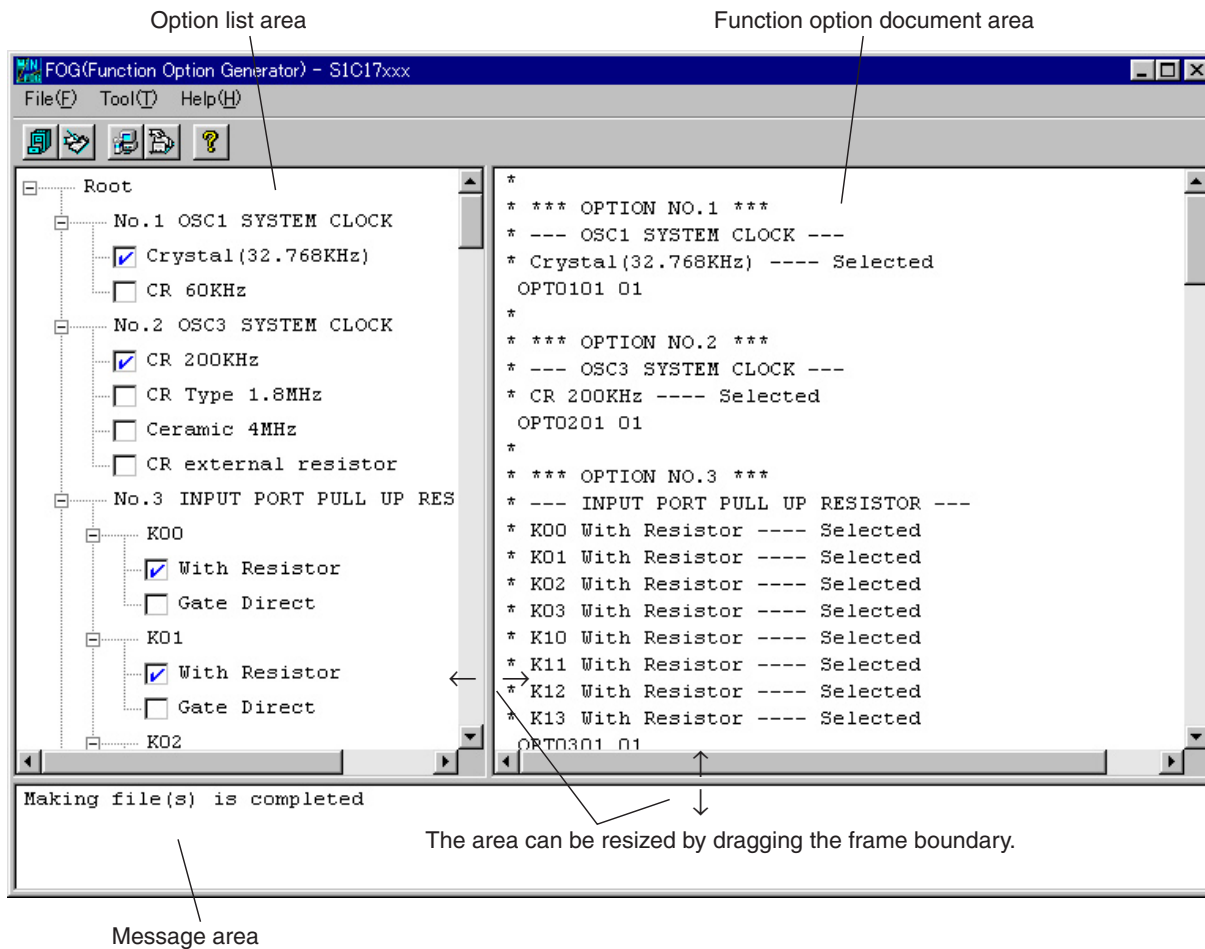


[FOG] Window (initial screen)



[FOG] Window (after reading the device information definition file)

## 11.9.4 Window



- \* The microcomputer model name on the title bar is the file name (not including the path and extension) of the device information definition file that has been read.
- \* The option list and the function option document vary with each type of microcomputer.

Figure 11.9.4.1 Window configuration

The [FOG] window is divided into three areas as shown above.

### Option list area

Lists mask options set in the device information definition file (S1C17xxx.ini). Use the check boxes in this area to select each option. A selected option has its check box marked by ✓.

### Function option document area

Displays the contents of selected options in the function option document format. The contents displayed in this area are output to the function option document file. When you change any selected item in the option list area, the display in this area is immediately updated.

### Message area

When you create a file by selecting [Generate] from the [Tool] menu or clicking the [Generate] button, this area displays a message showing the result of the selected operation.

## 11.9.5 Menus and Toolbar Buttons

This section explains each menu item and toolbar button.

### [File] menu



#### Open

Opens a function option document file. Use this menu command when correcting an existing file. The [Open] button has the same function.

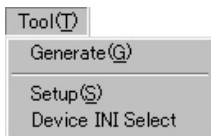


[Open] button

#### End

Terminates **winfog**.

### [Tool] menu



#### Generate

Creates a file according to the selected contents of the option list. The [Generate] button has the same function.



[Generate] button

#### Setup

Sets the date of creation, output file name and a comment included in the function option document file. The [Setup] button has the same function.



[Setup] button

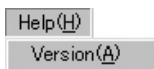
#### Device INI Select

Loads the device information definition file (S1C17xxx.ini). The [Device INI Select] button has the same function. This file must be loaded first before performing any operation with **winfog**.



[Device INI Select] button

### [Help] menu



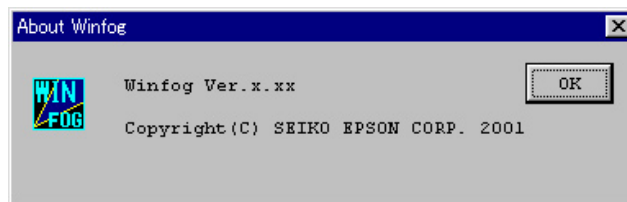
#### Version

Displays the version of **winfog**. The [Help] button has the same function.



[Help] button

The dialog box shown below appears. Click [OK] to close this dialog box.



## 11.9.6 Operation Procedure

The following shows the basic operation procedure.

### (1) Loading the device information definition file

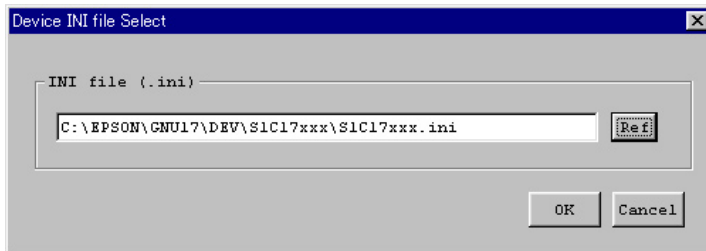
First, select a device information definition file (S1C17xxx.ini) and load it.

Select [Device INI Select] from the [Tool] menu or click the [Device INI Select] button.



[Device INI Select] button

The dialog box shown below appears. Enter a file name including the path in the text box or select a file by clicking the [Ref] button.



Click [OK], and the file is loaded. If the specified file exists and there is no problem with its contents, the option list and the function option document, which have both been set by default, are displayed in each area.

To stop loading the file, click [Cancel].

Once a device information definition file is selected, the same file is automatically loaded the next time you start **winfog**.

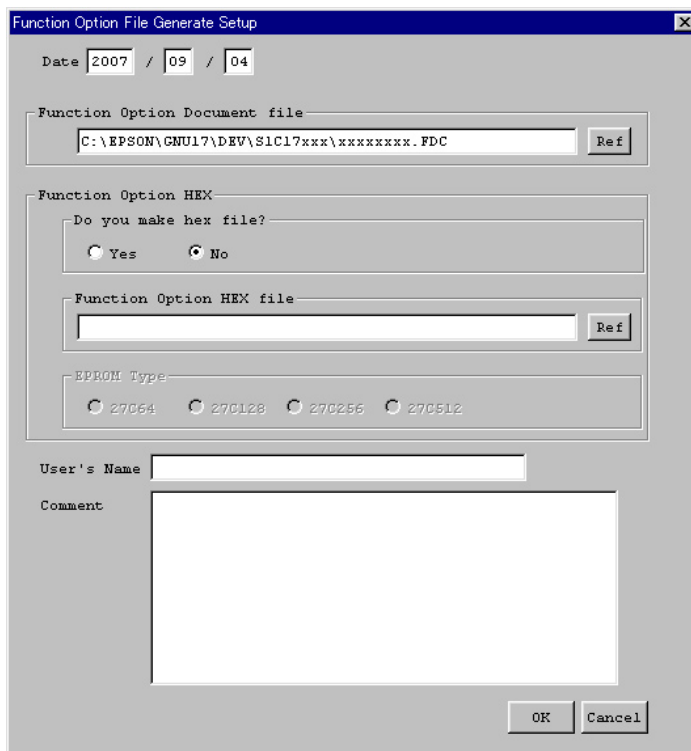
**Note:** When you load a device information definition file after setting up options, all settings are reset to the default state.

### (2) Setup

Select [Setup] from the [Tool] menu or click the [Setup] button to bring up the [Setup] dialog box. From this dialog box, select items and enter data.



[Setup] button



#### Date

Displays the current date. Change it as necessary.

#### Function Option Document file

Specify the function option document file name you want to create. The file name displayed by default can be modified. You can use the [Ref] button to look at other folders.

#### Function Option HEX

##### Do you make hex file?

Select whether to create a function option HEX file.

The S1C17 Family does not use a function option HEX file, so select "No".

#### EPROM Type

This option is not available for S1C17 Family microcomputers.



### User's Name

Enter your company name. You can enter up to 40 characters. You can use English letters, numbers, symbols, and spaces. The content entered here is recorded in the USER'S NAME field of the function option document file.

### Comment

Enter a comment. Up to 50 characters can be entered in one line. You can enter up to 10 lines. You can use English letters, numbers, symbols, and spaces. Use the [Enter] key to create a new line. All comments should include the following information:

- Place of business, your department or section
- Address, telephone number, and facsimile number
- Other: Technical information, etc.

The content entered here is recorded in the COMMENT field of the function option document file.

When you have finished entering the above necessary items, click [OK]. The setup contents are saved, and the dialog box is closed. The setup contents take effect immediately. If you click [Cancel], current settings will not be changed and the dialog box is closed.

**Notes:** • File name specification is subject to the following limitations:

1. The number of characters that can be used to specify a file name including the path is 2,048.
2. The file name itself (not including the extension) can be up to 15 characters, and the extension up to three characters.
3. The file name cannot begin with a hyphen (-), nor can the following symbols be used as part of directory names (folder names), file names, and extensions:  
/ : ; \* ? " < > |

- The symbols shown below cannot be used in the User's Name and Comment:

\$ \ | `

## (3) Selecting options

Select necessary options by clicking the corresponding check boxes in the option list. When you change any selection item in the option list area, the display in the function option document area is updated. Note that when you have loaded the device information definition file, the option list is placed in its default selection state.

For details about option specifications, refer to the Technical Manual available for each type of microcomputer.

## (4) Creating files

After selecting options, select [Generate] from the [Tool] menu or click the [Generate] button to create the files.



[Generate] button

The function option document file you specified from the [Setup] dialog box is created. When **winfog** has finished creating the files normally, it displays the message "Making file(s) is completed" in the message area. If an error occurs, an error message is displayed.

### (5) Correcting an existing document file

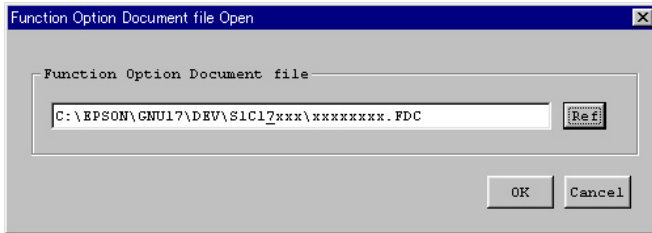
You can read an existing function option document file into winfog and correct it as necessary.

To read a file, select [Open] from the [File] menu or click the [Open] button.



[Open] button

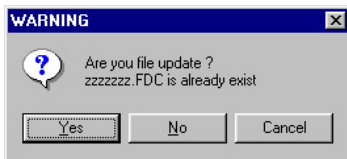
The dialog box shown below appears, so enter a file name including the path in the text box or select a file by clicking the [Ref] button.



Click [OK], and the file is loaded. If the specified file exists and there is no problem with its contents, the option list and the function option document areas are updated according to the contents of the file. To stop loading the file, click [Cancel].

Perform steps (2) to (4) to update the file.

If you select [Generate] without changing the file name, the message shown below is displayed asking you whether or not to overwrite the file. Click [Yes] to overwrite or [No] or [Cancel] to stop overwriting. Use the [Setup] dialog box to change the file name.



**Note:** The function option document file can be read only when the device information definition file has been loaded.

### (6) Quitting

To terminate **winfog**, select [End] from the [File] menu.

## 11.9.7 Error/Warning Messages

The error and warning messages of **winfog** are listed below. The "Dialog" in the Display column means that messages are displayed in the dialog box, and "Message" means that messages are displayed in the [FOG] window message area.

Table 11.9.7.1 List of error messages

Message	Description	Display
File name error	Number of characters in the file name or extension exceeds the limit.	Dialog
Illegal character	Prohibited characters have been entered.	Dialog
Please input file name	File name has not been entered.	Dialog
Can't open File : xxxx	File (xxxx) cannot be opened.	Dialog
INI file is not found	Specified device information definition file (.ini) does not exist.	Dialog
INI file does not include FOG information	Specified device information definition file (.ini) does not contain function option information.	Dialog
Function Option document file is not found	Specified function option document file does not exist.	Dialog
Function Option document file does not match INI file	Contents of the specified function option document file do not match device information definition file (.ini).	Dialog
A lot of parameter	Too many command line parameters are specified.	Dialog
Making file(s) is completed [xxxx is no data exist]	Finished creating the file, but the created file (xxxx) does not contain any data.	Message
Can't open File: xxxx Making file(s) is not completed	File (xxxx) cannot be opened when executing Generate.	Message
Can't write File: xxxx Making file(s) is not completed	File (xxxx) cannot be written when executing Generate.	Message

Table 11.9.7.2 Warning message

Message	Description	Display
Are you file update? xxxx is already exist	Overwrite confirmation message (Specified file already exists.)	Dialog

## 11.9.8 Sample Output File

**Note:** Option and other configurations vary with each type of microcomputer.

### Example of a function option document file

```

* S1C17xxx FUNCTION OPTION DOCUMENT Vx.xx ← Version
*
* FILE NAME      zzzzzzzz.FDC           ← File name (specified by [Setup])
* USER'S NAME   SEIKO EPSON CORPORATION ← User name (specified by [Setup])
* INPUT DATE    yyyy/mm/dd             ← Date of creation (specified by [Setup])
* COMMENT       SAMPLE DATA           ← Comment (specified by [Setup])
*
* *** OPTION NO.1 ***                  ← Option number
* --- OSC1 SYSTEM CLOCK ---           ← Option name
* Crystal(32.768KHz) ---- Selected    ← Selected specification
OPT0101 01                             ← Mask data
*
* *** OPTION NO.2 ***
* --- OSC3 SYSTEM CLOCK ---
* CR 200KHz ---- Selected
OPT0201 01
*
* *** OPTION NO.3 ***
* --- INPUT PORT PULL UP RESISTOR ---
* K00 With Resistor ---- Selected
* K01 With Resistor ---- Selected
* K02 With Resistor ---- Selected
* K03 With Resistor ---- Selected
* K04 With Resistor ---- Selected
* K05 With Resistor ---- Selected
* K06 With Resistor ---- Selected
* K07 With Resistor ---- Selected
OPT0301 01
OPT0302 01
OPT0303 01
OPT0304 01
OPT0305 01
OPT0306 01
OPT0307 01
OPT0308 01
*
* *** OPTION NO.4 ***
* --- OUTPUT PORT OUTPUT SPECIFICATION ---
* R00 Complementary ---- Selected
* R01 Complementary ---- Selected
* R02 Complementary ---- Selected
* R03 Complementary ---- Selected
OPT0401 01
OPT0402 01
OPT0403 01
OPT0404 01
*
*
*
* *** OPTION NO.8 ***
* --- SOUND GENERATOR POLARITY ---
* NEGATIVE ---- Selected
OPT0801 01
*EOF                                     ← End mark

```

## 11.10 winmdc.exe

### 11.10.1 Outline of winmdc

The Mask Data Checker **winmdc** is the software tool for checking the format of each generated file and creating the files necessary to generate mask patterns. The **winmdc** checks the built-in ROM data file generated by **objcopy**, **moto2ff** and **sconv32**, and the function option document file generated by **winfog**.

The **winmdc** also has a function for restoring the created mask data file into the original file format.

### 11.10.2 Input/Output Files

Figure 11.10.2.1 shows the input/output files of **winmdc**.

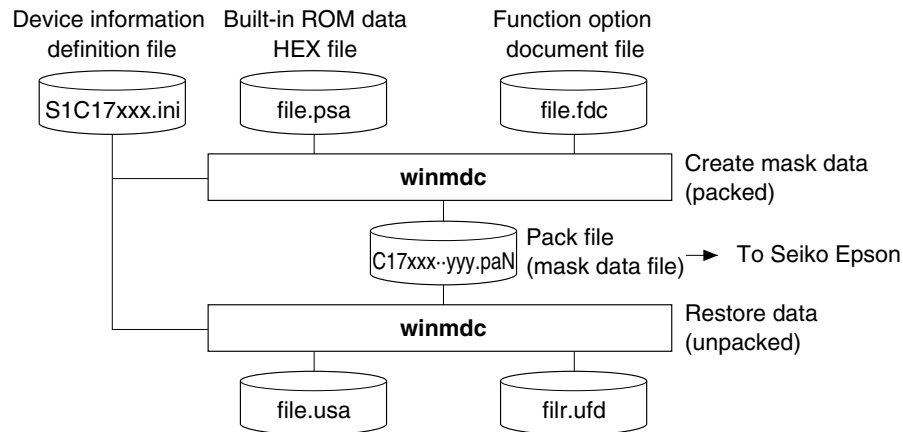


Figure 11.10.2.1 Input/output files of **winmdc**

#### Device information definition file

File format: Text format file

File name: S1C17xxx.ini

Description: This file contains option lists for various types of microcomputers and other information. Always be sure to use the files presented by Seiko Epson. This file is effective for only the type of microcomputer indicated by the file name. Do not modify the contents of the file or use the file in other types of microcomputers.

#### Built-in ROM data HEX file

File format: Motorola S2 format file

File name: <file name>.psa

Description: This is the built-in ROM data file in Motorola S2 format. This file is created by **objcopy**, **moto2ff** and **sconv32**. The unused areas in the built-in ROM are filled with 0xff.

#### Function option document file

File format: Text format file

File name: <file name>.fdc

Description: This is a text format file in which the contents of selected function options are stored. This file is created by function option generator **winfog**.

#### Pack file

File format: Text format file

File name: C17xxx-yyy.paN (N = 0 and over)

Description: This is a text format file which contains the above data files combined into one. We would like to have this file presented to Seiko Epson as the mask data file. Seiko Epson will create the mask patterns for the IC from this mask data file.

\* The "xxx-" in the file name denotes the model name of a microcomputer. The "yyy" part of the file name represents the custom code of each customer. Enter the code from Seiko Epson here. For the <file name> part, any given file name can be specified.

### 11.10.3 Starting Up

#### Startup from Explorer



winmdc.exe

Double-click on the **winmdc.exe** icon or select **winmdc** from the start menu.

If the device information definition file (S1C17xxx.ini) was loaded into your computer during a previous execution, **winmdc** automatically reads the same file as it starts.

Alternatively, drag the device information definition file icon into the **winmdc.exe** icon to start **winmdc**, which will then read the device information definition file.

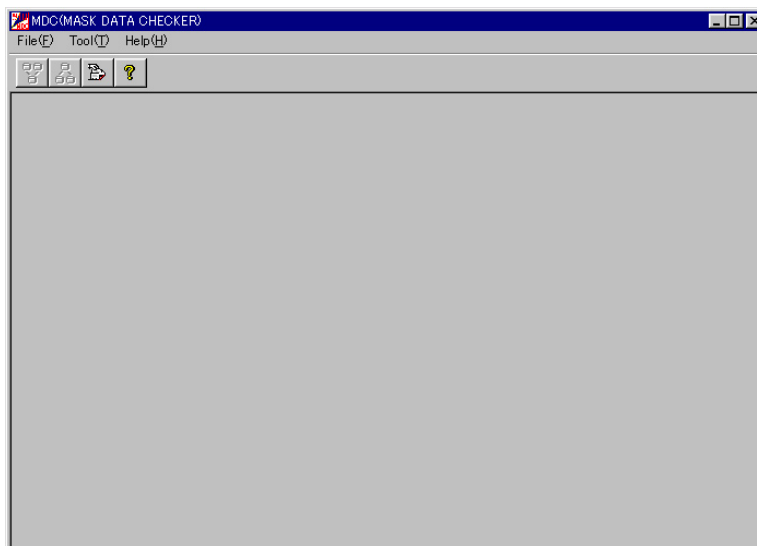
#### Startup by command input

You can also start **winmdc** from the command prompt by entering the command shown below.

```
>winmdc [S1C17xxx.ini]
```

You can specify the device information definition file (S1C17xxx.ini) as a command option. (You can also specify a path.) When you specify the Device information definition file here, **winmdc** reads it as it starts. This specification can be omitted.

When **winmdc** starts, it displays the [MDC] window.



[MDC] Window (initial screen)

- \* The microcomputer model name on the title bar is the file name (not including the path and extension) of the device information definition file that has been read.
- \* The [Pack] and [Unpack] buttons on the tool bar are enabled when the device information definition file is read.

## 11.10.4 Menus and Toolbar Buttons

This section explains each menu item and toolbar button.

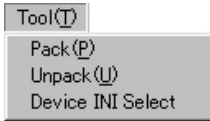
### [File] menu



#### End

Terminates **winmdc**.

### [Tool] menu



#### Pack

Packs the ROM data file and option document file to create a mask data file for presentation to Seiko Epson. The [Pack] button has the same function.



[Pack] button

#### Unpack

Restores files in the original format from a packed file. The [Unpack] button has the same function.



[Unpack] button

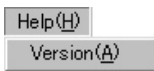
#### Device INI Select

Loads the device information definition file (S1C17xxx.ini). The [Device INI Select] button has the same function. This file must be loaded first before performing any operation with **winmdc**.



[Device INI Select] button

### [Help] menu



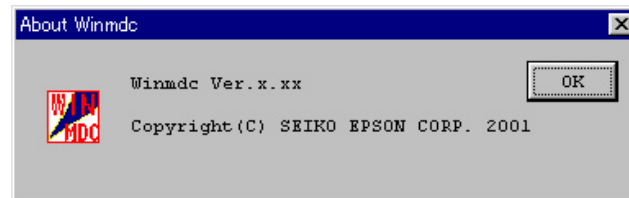
#### Version

Displays the version of **winmdc**. The [Help] button has the same function.



[Help] button

The dialog box shown below appears. Click [OK] to close this dialog box.



## 11.10.5 Operation Procedure

The following shows the basic operation procedure.

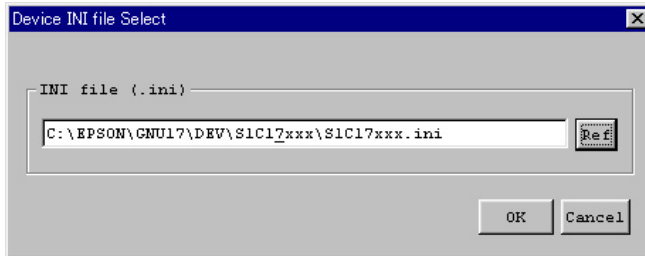
### (1) Loading the device information definition file

First, select a device information definition file (S1C17xxx.ini) and load it.

Select [Device INI Select] from the [Tool] menu or click the [Device INI Select] button.

 [Device INI Select] button

The dialog box shown below appears. Enter a file name including the path in the text box or select a file by clicking the [Ref] button.



Click [OK], and the file is loaded. If the specified file exists and there is no problem with its contents, the set-up items in **winmdc** are initialized with the loaded device information.

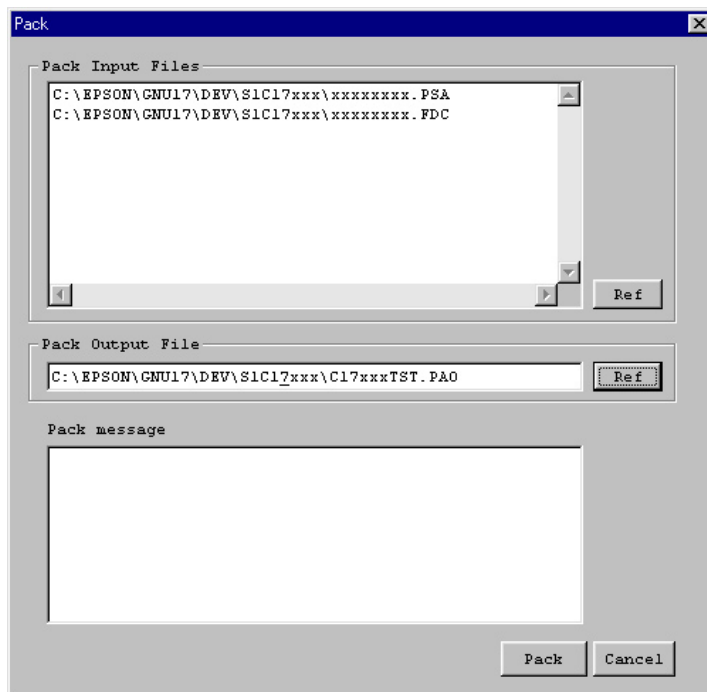
To stop loading the file, click [Cancel].

Once a device information definition file is selected, the same file is automatically loaded the next time you start **winmdc**.

### (2) Packing

1. Select [Pack] from the [Tool] menu or click the [Pack] button on the tool bar to bring up the [Pack] dialog box.

 [Pack] button





2. Select the files to be entered.  
[Pack Input Files] lists the files of the type specified in the device information definition file by their default file names. If the data files to be entered are represented by different names in this list, replace the file names following the procedure below.

- a. Select a file name to be changed by clicking on it from the list box.
- b. Click the [Ref] button and select the data file to be entered.

Do this for all files listed.

When replacing files, take care not to mistake one file type (extension) for another. If the type of input file is erroneous, an error will result during file packing.

3. Setting output file names.

In the [Pack Output File] text box, specify a pack file name in which you want the mask data to be output. The file name displayed by default can be modified. You can use the [Ref] button to look at other folders.

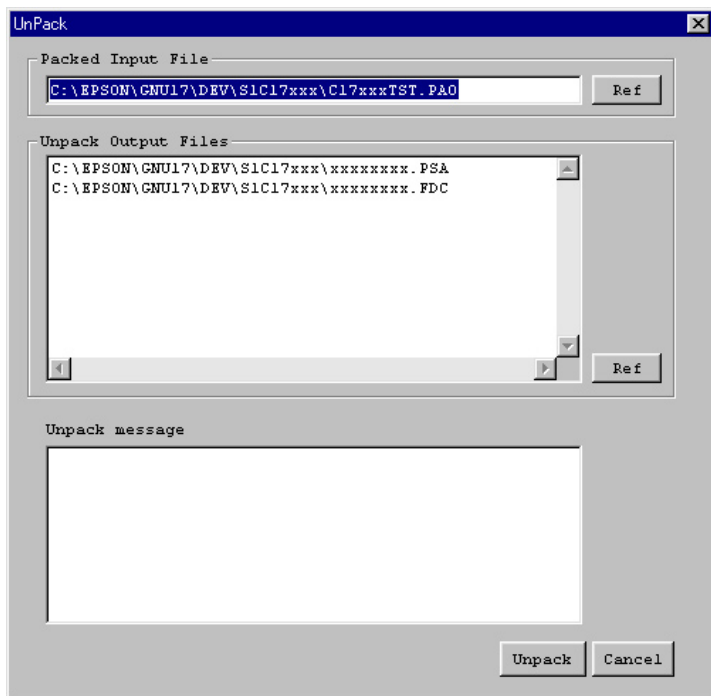
Make sure the extension of the output file name is ".pa0". If after presenting data to Seiko Epson, you present new data due to program bugs or any other reason, increase the number in the last digit of the extension in increments of one. For example, the extension of the second file presented should be "c17xxx·yyy.pa1".

**Note:** File name specification is subject to the following limitations:

1. The number of characters that can be used to specify a file name including the path is 2,048.
2. The file name itself (not including the extension) can be up to 15 characters, and the extension up to three characters.
3. The file name cannot begin with a hyphen (-), nor can the following symbols be used as part of directory names (folder names), file names, and extensions:  
/ : , ; \* ? " < > |
4. Click the [Pack] button to execute packing.  
When **winmdc** has completed packing, it displays a message "Packing completed!" in the [Pack message] text box. If an error has occurred, an error message is displayed.
5. Click the [Cancel] button to close the dialog box.  
Alternatively, you can click the [Cancel] button to quit **winmdc** before it executes packing.

### (3) Unpacking

1. Select [Unpack] from the [Tool] menu or click the [Unpack] button on the tool bar to bring up the [Unpack] dialog box.



2. Select the file you want to unpack.  
In the [Packed Input File] text box, specify the pack file name you want to enter. Use the names displayed by default to specify this file name after changing one, or select another file using the [Ref] button.
3. Select the output file name.  
[Unpack Output Files] lists the files of the type specified in the device information definition file by their default file names. Modify the file name displayed by the following procedure.
  - a. Click in the list box to select the file name to be modified.
  - b. Click the [Ref] button to select another folder, and then enter a file name. Modify all the listed file names.  
The extensions cannot be changed.
4. Click the [Unpack] button to execute unpacking.  
When **winmdc** has completed unpacking, it displays a message "Unpacking completed!" in the [Unpack message] text box. If an error has occurred, an error message is displayed.
5. Click the [Cancel] button to close the dialog box.  
Alternatively, you can click the [Cancel] button to quit **winmdc** before it executes unpacking.

### (4) Quitting

To terminate **winmdc**, select [End] from the [File] menu.

## 11.10.6 Error Messages

The error messages of **winmdc** are listed below. The "Dialog" in the Display column means that messages are displayed in the dialog box, and "Message" means that messages are displayed in the message area of the [Pack] or [Unpack] dialog box.

Table 11.10.6.1 List of I/O error messages

Message	Description	Display
File name error	Number of characters in the file name or extension exceeds the limit.	Dialog
Illegal character	Prohibited characters have been entered.	Dialog
Please input file name	File name has not been entered.	Dialog
INI file is not found	Specified device information definition file (.ini) does not exist.	Dialog
INI file does not include MDC information	Specified device information definition file (.ini) does not contain MDC information.	Dialog
Can't open file : xxxx	File (xxxx) cannot be opened.	Dialog
Can't write file: xxxx	File (xxxx) cannot be written.	Dialog

Table 11.10.6.2 List of ROM data error messages

Message	Description	Display
Hex data error: Not S record.	Data does not begin with "S."	Message
Hex data error: Data is not sequential.	Data is not listed in ascending order.	Message
Hex data error: Illegal data.	Invalid character is included.	Message
Hex data error: Too many data in one line.	Too many data entries exist in one line.	Message
Hex data error: Check sum error.	Checksum does not match.	Message
Hex data error: ROM capacity over.	Data is large. (Greater than ROM size)	Message
Hex data error: Not enough the ROM data.	Data is small. (Smaller than ROM size)	Message
Hex data error: Illegal start mark.	Start mark is incorrect.	Message
Hex data error: Illegal end mark.	End mark is incorrect.	Message
Hex data error: Illegal comment.	Model name shown at the beginning of data is incorrect.	Message

Table 11.10.6.3 List of function option data error messages

Message	Description	Display
Option data error : Illegal model name.	Model name is incorrect.	Message
Option data error : Illegal version.	Version is incorrect.	Message
Option data error : Illegal option number.	Option No. is incorrect.	Message
Option data error : Illegal select number.	Selected option number is incorrect.	Message
Option data error : Mask data is not enough.	Mask data is insufficient.	Message
Option data error : Illegal start mark.	Start mark is incorrect.	Message
Option data error : Illegal end mark.	End mark is incorrect.	Message

## 11.10.7 Sample Output File

**Note:** The configuration and contents of data vary with each type of microcomputer.

### Example of a pack file (mask data file)

```

*
* S1C17xxx MASK DATA VER x.xx           ← Version
*
¥ROM1                                     ← Built-in ROM HEX data start mark
S1C17xxxyyy PROGRAM ROM                 ← Model name
S224000000.....
:      :      :      :      :
S804000000FB                             "xxxxxxx.psa"
S224000000.....
:      :      :      :      :
S804000000FB
¥END                                     ← Built-in ROM HEX data end mark
¥OPTION1                                 ← Function option start mark
* S1C17xxx FUNCTION OPTION DOCUMENT V x.xx ← Model name/version
*
* FILE NAME      zzzzzzzz.FDC
* USER'S NAME    SEIKO EPSON CORPORATION
* INPUT DATE      yyyy/mm/dd
* COMMENT         SAMPLE DATA
*
* *** OPTION NO.1 ***
* --- OSC1 SYSTEM CLOCK ---
* Crystal(32.768KHz) ---- Selected
OPT0101 01
:      :      :      :      :
OPTnn01 01
*EOF
¥END                                     ← Function option end mark

```

S1C17 Family C Compiler Package

# Quick Reference



# EPSON

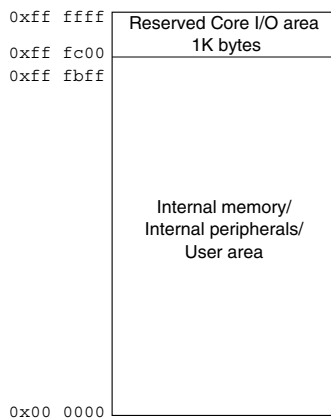
CMOS 16-bit Single Chip Microcomputer  
S1C17 Family C Compiler Package

## Quick Reference for Development

### Memory Map and Trap Table (S1C17 Core)

S1C17 Core

#### Memory Map



#### Trap Table

No.		Vector address
0 (0x00)	Reset	TTBR + 0x00
1 (0x01)	Address misaligned interrupt	TTBR + 0x04
2 (0x02)	NMI	TTBR + 0x08
3 (0x03)	Maskable external interrupt 3	TTBR + 0x0c
⋮	⋮	⋮
31 (0x1f)	Maskable external interrupt 31	TTBR + 0x7c

TTBR: Trap table start address  
(Can be read from address 0xffff80.)

### Registers (S1C17 Core)

S1C17 Core

#### General-purpose Registers (8)

23		0
	R7	
	R6	
	R5	
	R4	
	R3	
	R2	
	R1	
	R0	

#### Special Registers (3)

23		0	
	PC		Program counter
23		0	
	SP		Stack pointer
7		0	
	PSR		Processor status register

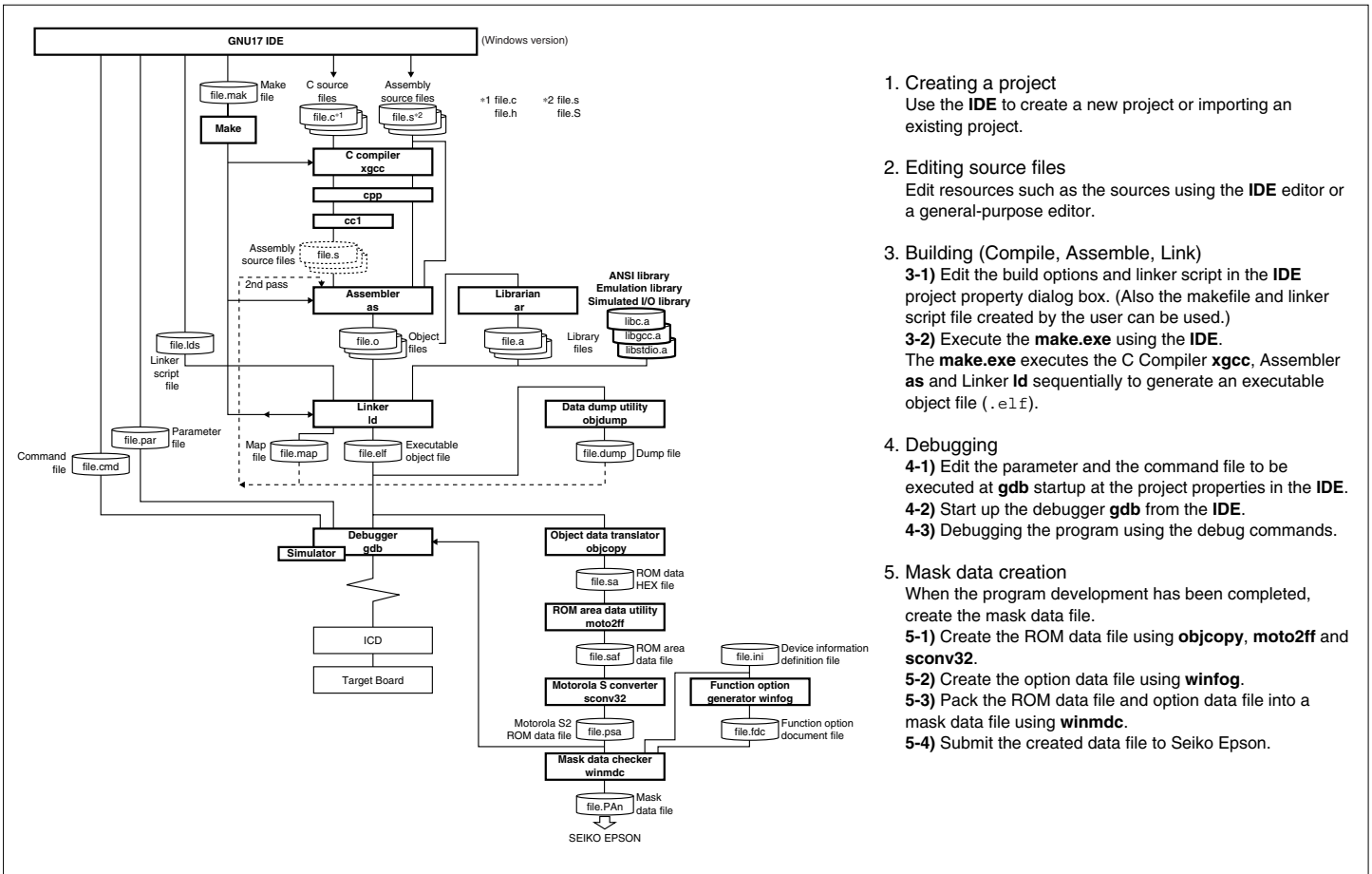
#### PSR

7	6	5	4	3	2	1	0	
			IL[2:0]	IE	C	V	Z	N
Initial value	0	0	0	0	0	0	0	0

- IL[2:0]: Interrupt level (0–7: Enabled interrupt level)
- IE: Interrupt enable (1: Enabled, 0: Disabled)
- Z: Zero flag (1: Zero, 0: Non zero)
- N: Negative flag (1: Negative, 0: Positive)
- C: Carry flag (1: Carry/borrow, 0: No carry)
- V: Overflow flag (1: Overflow, 0: Not overflown)

## Software Development Flowchart

## Development Tools



### 1. Creating a project

Use the **IDE** to create a new project or importing an existing project.

### 2. Editing source files

Edit resources such as the sources using the **IDE** editor or a general-purpose editor.

### 3. Building (Compile, Assemble, Link)

**3-1** Edit the build options and linker script in the **IDE** project property dialog box. (Also the makefile and linker script file created by the user can be used.)

**3-2** Execute the **make.exe** using the **IDE**.

The **make.exe** executes the C Compiler **xgcc**, Assembler **as** and Linker **ld** sequentially to generate an executable object file (.elf).

### 4. Debugging

**4-1** Edit the parameter and the command file to be executed at **gdb** startup at the project properties in the **IDE**.

**4-2** Start up the debugger **gdb** from the **IDE**.

**4-3** Debugging the program using the debug commands.

### 5. Mask data creation

When the program development has been completed, create the mask data file.

**5-1** Create the ROM data file using **objcopy**, **moto2ff** and **sconv32**.

**5-2** Create the option data file using **winfog**.

**5-3** Pack the ROM data file and option data file into a mask data file using **winmdc**.

**5-4** Submit the created data file to Seiko Epson.

## GNU17 IDE (1)

## Development Tools

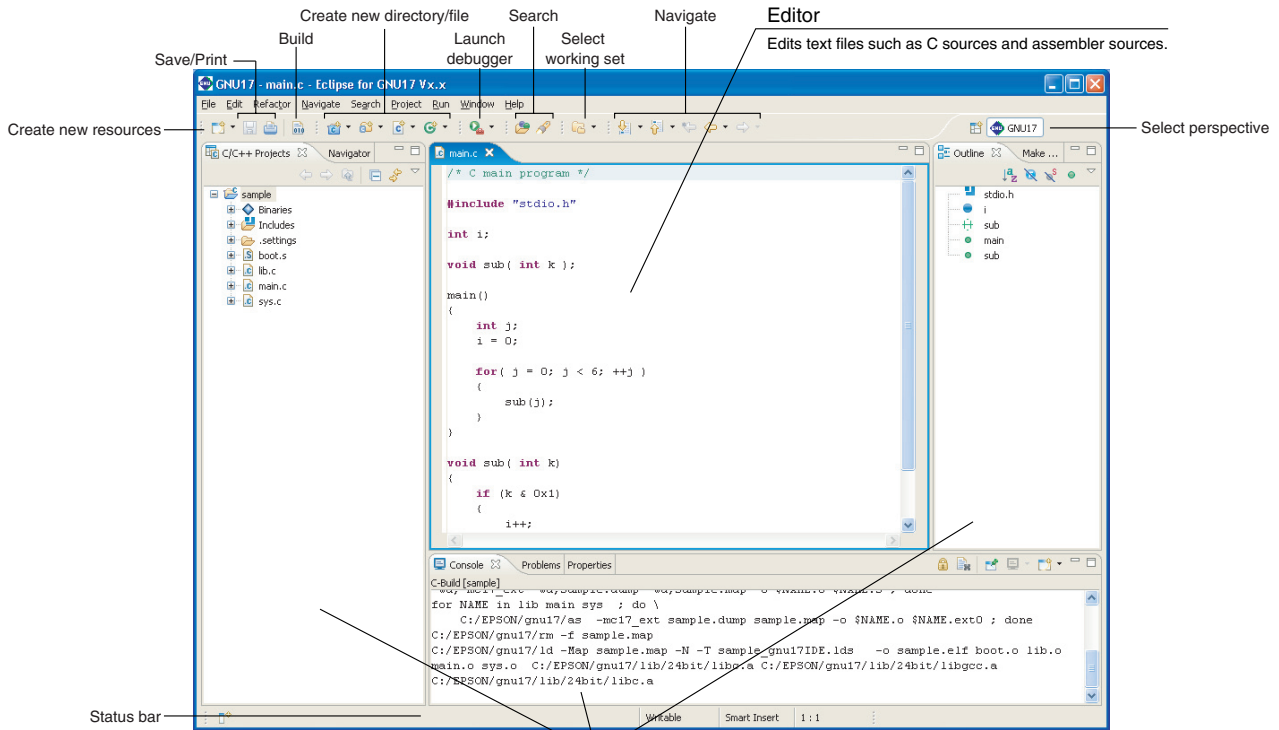
### Overview



The **GNU17 IDE** provides an integrated development environment that allows the user to easily develop software with the S1C17 Family C Compiler Package (S5U1C17001C).

### Start-up Command

eclipse



### Views

Displays various information by contents. To open a closed view, select it from [Show View] on the [Window] menu.



## Menu Bar

## [File] menu

File	
New	Alt+Shift+N
Open File...	
Close	Ctrl+W
Close All	Ctrl+Shift+W
Save	Ctrl+S
Save As...	
Save All	Ctrl+Shift+S
Revert	
Move...	
Rename...	F2
Refresh	F5
Convert Line Delimiters To	
Print...	Ctrl+P
Switch Workspace...	
Import...	
Export...	
Properties	Alt+Enter
1 main.c [sample]	
Exit	

## [Edit] menu

Edit	
Undo Typing	Ctrl+Z
Redo Typing	Ctrl+Y
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Delete	Delete
Select All	Ctrl+A

**New (Alt+Shift+N)** Creates a new project, a new file and a new directory.

**Open File...** Choose a file to be opened with the editor.

**Close (Ctrl+W)** Closes the current active file.

**Close All (Ctrl+Shift+W)** Closes all files open in the editor.

**Save (Ctrl+S)** Saves changes made in the current file.

**Save As...** Saves the current active file under another name.

**Save All (Ctrl+Shift+S)** Saves all open files.

**Revert** Discards any changes made in the current active file, reverting to the previously saved version.

**Move...** Moves the file or directory selected in the [C/C++ Projects]/[Navigator] view to a different location.

**Rename... (F2)** Places the file or directory selected in the [C/C++ Projects]/[Navigator] view in editing mode (allowing renaming of the file or directory).

**Refresh (F5)** Updates the displayed content of the [C/C++ Projects]/[Navigator] view.

**Convert Line Delimiters To** Selects a line delimiting character.

**Print... (Ctrl+P)** Prints the current active file.

**Switch Workspace...** Selects another workspace.

**Import...** Adds an existing project or source file to the current workspace or project.

**Export...** Writes the file in the current project out to another directory.

**Properties (Alt+Enter)** Displays or edits properties of the project, file, or directory currently selected in the [C/C++ Projects]/[Navigator] view.

**Exit** Closes the IDE.

**Undo Typing (Ctrl+Z)** Undoes the most recent operation performed.

**Redo Typing (Ctrl+Y)** Repeats the last operation canceled by [Undo Typing].

**Cut (Ctrl+X)** Cuts the selected string/file/directory.

**Copy (Ctrl+C)** Copies a selected string/file/directory.

**Paste (Ctrl+V)** Pastes the copied content from the clipboard.

**Delete (Delete)** Deletes the selected string/file/directory.

**Select All (Ctrl+A)** Selects all of the contents in currently active document in the editor.

## Menu Bar

## [Edit] menu

Edit	
Undo Typing	Ctrl+Z
Redo Typing	Ctrl+Y
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Delete	Delete
Select All	Ctrl+A
Find/Replace...	Ctrl+F
Find Next	Ctrl+K
Find Previous	Ctrl+Shift+K
Incremental Find Next	Ctrl+J
Incremental Find Previous	
Add Bookmark...	
Add Task...	
Word Completion	Alt+/ Ctrl+I
Shift Right	Ctrl+I
Shift Left	Ctrl+Shift+I
Content Assist	Ctrl+Space
Add Include	Ctrl+Shift+N
Format	Ctrl+Shift+F
Open On Selection	
Set Encoding...	

## [Refactor] menu

Refactor	
Undo	Alt+Shift+Z
Redo	Alt+Shift+Y
Rename...	Alt+Shift+R

**Find/Replace... (Ctrl+F)** Finds and replaces a string in the editor.

**Find Next (Ctrl+K)** Jumps to the next instance of a search string.

**Find Previous (Ctrl+Shift+K)** Jumps back to the previous instance of a search string.

**Incremental Find Next (Ctrl+J)** Performs incremental search backward from the current position in the editor.

**Incremental Find Previous** Performs incremental search forward from the current position in the editor.

**Add Bookmark...** Registers a line in an active document in the editor at the current cursor position as a bookmark.

**Add Task...** Registers the line at the current cursor position in an active document in the editor as a task (memorandum).

**Word Completion (Alt+/  
Ctrl+I)** Inserts a word that begins with the letters being entered in the editor to the current position.

**Shift Right (Ctrl+I)** Shifts the beginning of a line one tab position to the right.

**Shift Left (Ctrl+Shift+I)** Shifts the beginning of a line one tab position to the left.

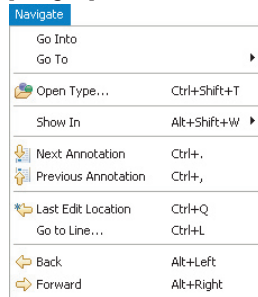
**Content Assist (Ctrl+Space)** Input assistance for a reserved word or template in a C source.

**Set Encoding...** Selects the text-encoding format.

**Undo (Alt+Shift+Z)** Undoes the most recent rename operation.

**Redo (Alt+Shift+Y)** Repeats the rename operation canceled by [Undo].

**Rename... (Alt+Shift+R)** Changes the selected function or member name, all instances including these in other locations of the source.

**Menu Bar****[Navigate] menu**

**Go Into** Changes display of the [C/C++ Projects]/[Navigator] view to display the content of just the currently selected directory.

**Go To** Navigates the display history of the [C/C++ Projects]/[Navigator] view.

**Show In (Alt+Shift+W)** Selects a view to highlight the resource that includes the selected function name, variable name, or type.

**Next Annotation (Ctrl+.)** Selects the next item the list displayed in the [Problems] or the [Search] view.

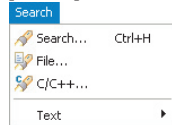
**Previous Annotation (Ctrl+.)** Selects the previous item the list displayed in the [Problems] or the [Search] view.

**Last Edit Location (Ctrl+Q)** Jumps to the last edited position in the editor.

**Go to Line... (Ctrl+L)** Jumps to the position in the active document indicated by the specified line number.

**Back (Alt+Left)** Returns to any position in the document just referenced or edited.

**Forward (Alt+Right)** Reverts the display traced back by [Back] above to the next recent state.

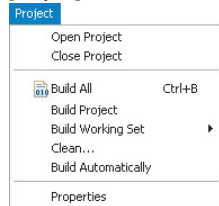
**[Search] menu**

**Search... (Ctrl+H)** Displays a [Search] dialog box that lets the user search for a file or C.

**File...** Searches for a file containing the specified string.

**C/C++...** Searches for C source containing the specified string.

**Text** Searches a string from the specified range.

**[Project] menu**

**Open Project** Opens the closed project currently selected in the [C/C++ Projects]/[Navigator] view.

**Close Project** Closes the project currently selected in the [C/C++ Projects]/[Navigator] view.

**Build All (Ctrl+B)** Executes a build process on all projects open in the [C/C++ Projects]/[Navigator] view.

**Build Project** Executes a build process on the project currently selected in the [C/C++ Projects]/[Navigator] view.

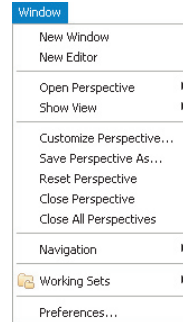
**Build Working Set** Executes a build process on the resources included in a specified working set.

**Clean...** Executes a clean or a rebuild.

**Properties** Displays a [Properties] dialog box that lets the user display or edit properties of the project selected in the [C/C++ Projects] or [Navigator] view.

**Menu Bar****[Run] menu**

**External Tools** Starts the **gdb** debugger or an external tool.

**[Window] menu**

**New Window** Opens a new window.

**New Editor** Opens the currently edited file with the new editor tab.

**Open Perspective** Opens the perspective.

**Show View** Opens a view.

**Customize Perspective...** Changes settings for the current perspective.

**Save Perspective As...** Saves settings for the current perspective under another name.

**Reset Perspective** Restores the perspective to the default state.

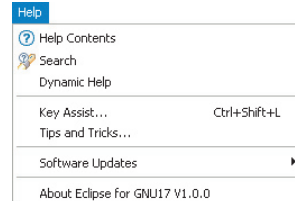
**Close Perspective** Closes the currently active perspective.

**Close All Perspectives** Closes all loaded perspectives.

**Navigation** Navigates the editor or view.

**Working Sets** Creates, edits, or deletes a working set.

**Preferences...** Customizes the IDE environment.

**[Help] menu**

**Help Contents** Displays Help files.

**Search** Displays a search view for help topics.

**Dynamic Help** Displays the help topic related to the view currently activated.

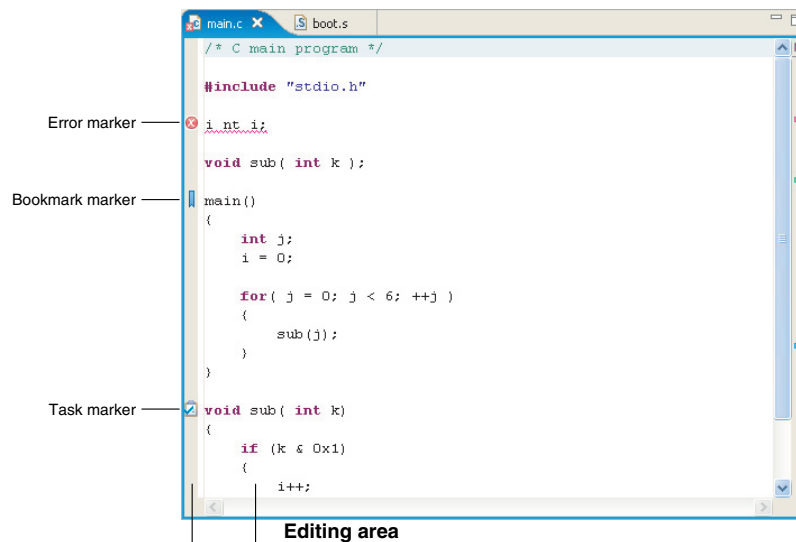
**Key Assist... (Ctrl+Shift+L)** Displays the list of currently available menu commands.

**Software Updates** Installs an updater, updates, plug-ins, etc. for software management.

**About Eclipse for GNU17 Vx.x** Shows IDE version information and detailed information on plug-ins, etc.

### Editor Area

The area of the editor where you edit source code. The IDE opens the C editor or the assembler editor according to the file type to be edited.



### Marker bar

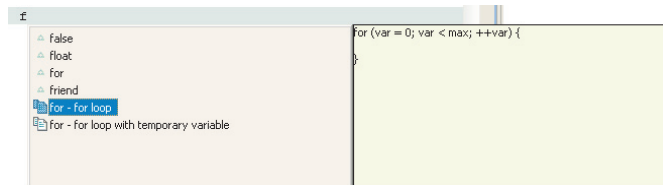
The marker bar shows the line in error and the markers indicating a bookmark, a line in which a task is set, etc. Markers are displayed on the left edge of the corresponding line.

### Editing area

Edit sources in this area. While a C source being edited, the content assist function shown below can be used by pressing the [Ctrl] + [Space] keys.

### Overview ruler

The overview ruler shows the position in error and the position at which a bookmark or task is set by a square symbol. Click a marker to go to that position.

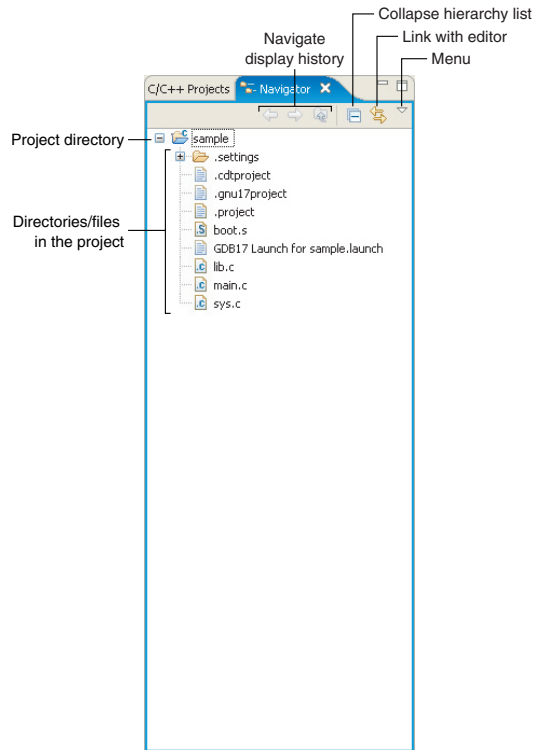
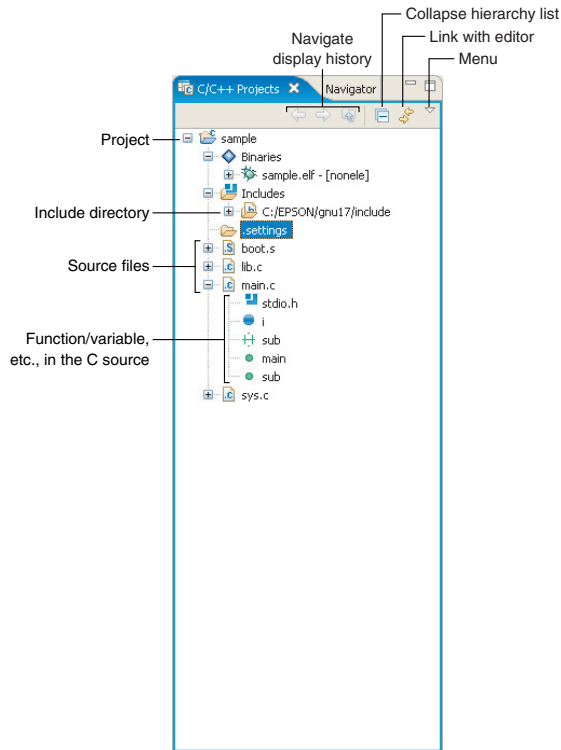


**[C/C++ Projects] View**

Lists the projects present in the workspace along with the C and assembler sources, include files, and generated execution format object files included in these projects. (Select the type of file to be displayed using [Filters...] from the view menu.) The function names and global variable names, etc. in the C source can also be displayed. Before editing a project or source or performing other operations, be sure to select the desired project or source here.

**[Navigator] View**

Lists the directories and files present in the workspace. (The type of file to be displayed can be selected using [Filters...] from the view menu.) Before editing a project or source or performing other operations, select the desired project or source here.



**[Outline] View**

Shows the functions and global variables that are written in the C source being displayed in the editor. Clicking on one of these items allows you to jump to the position in the editor at which the function or variable is written. While an assembler source is being displayed, no information is shown in this view.

Sort in alphabetical order

Display/hide fields, static members, members other than public

Menu

**Icons**

- Project
- Binary container
- Executable format file
- Include container
- Include folder
- Header file
- Source folder
- C source file
- Assembler source file
- Text file, etc.
- Object file
- Include
- Variable
- Function
- Structure (struct)
- Member variable
- Union (union)
- Enumeration type (enum)
- Enumerator
- Function definition (prototype declaration)
- Macro-definition
- Type definition (typedef)

**[Make Target] View**

When using a makefile you created, specify a target to execute.

Navigate

Execute a make process at a selected target

Hide the folders in which no makefile exists.

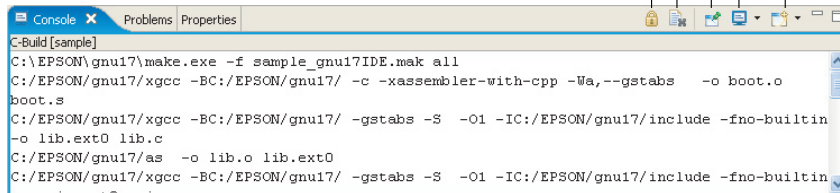
Project directory

Targets in the project

**[Console] View**

Displays the executed command line or the messages output by the GNU17 tools.

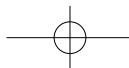
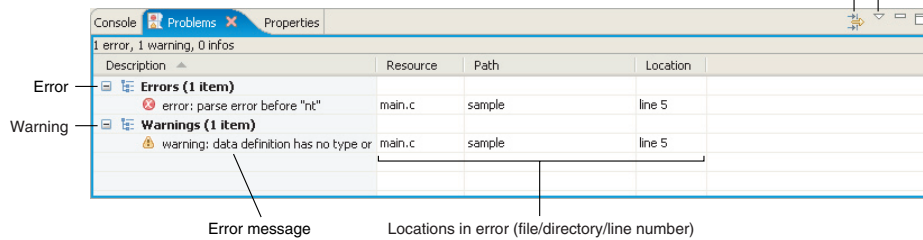
Clear the contents displayed  
Automatic scroll lock  
Enable other view while the [Console] view is displaying messages.  
Switching [Console]  
Opens a new [Console].



**[Problems] View**

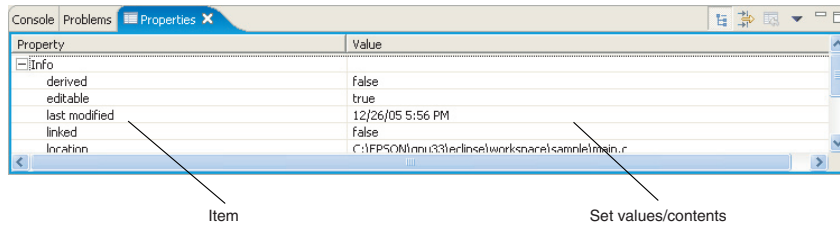
Shows the errors that occurred during a build operation. For errors in the source file, you can jump to the corresponding spot in the editor that is in error by clicking on an error message here.

Display filter  
Menu



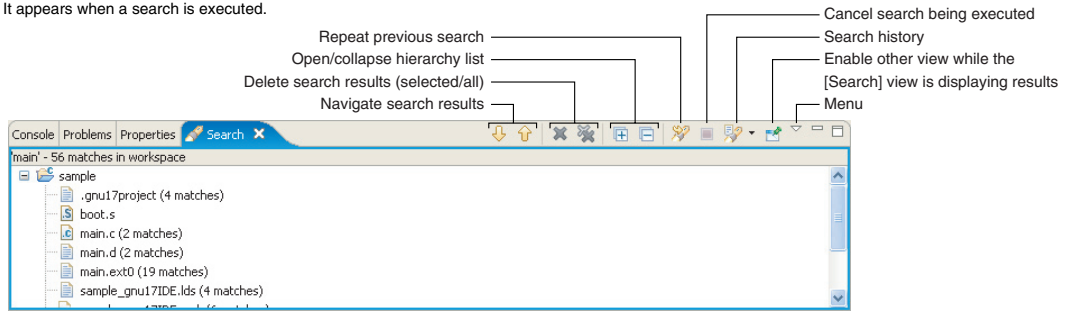
**[Properties] View**

Displays information on the resource or member currently selected in the [C/C++ Projects], the [Navigator], or the [Outline] view.



**[Search] View**

Shows the result of a search that was performed using the [Search] dialog box. This view in the initial IDE configuration is not displayed. It appears when a search is executed.



**[Bookmarks] View**

Shows the bookmarks registered in the editor, letting you jump to a bookmark or delete a bookmark.

Description	Resource	Path	Location
Sub Function	main.c	sample	line 9
Sub Function	main.c	sample	line 20

Bookmark name/description

Bookmark set locations (file/directory/line number)

**[Tasks] View**

Shows the tasks (To-Do) registered in the editor, letting you jump to or delete a task.

Description	Resource	Path	Location
<input checked="" type="checkbox"/> Add Sub1 Function	main.c	sample	line 19
<input type="checkbox"/> Modify Statement	main.c	sample	line 16
<input type="checkbox"/> Check Performance	main.c	sample	line 25

Completion mark (check) display

Priority

- High
- (blank) Normal
- Low

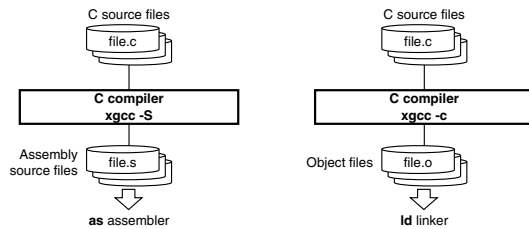
Description for tasks

Task set locations (file/directory/line number)



**Outline**

This tool is made based on GNU C Compiler and is compatible with ANSI C. This tool invokes **cpp.exe** and **cc1.exe** sequentially to compile C source files to the assembly source files for the S1C17 Family. It has a powerful optimizing ability that can generate minimized assembly codes. The **xgcc.exe** can also invoke the **as.exe** assembler to generate object files.

**Flowchart****Start-up Command**

**xgcc** <options> <filename>

<filename> C source file name

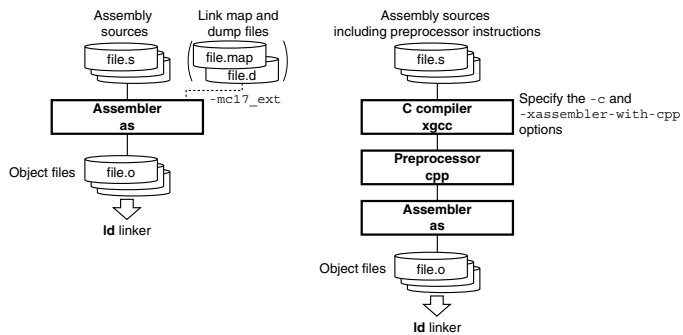
Example: `xgcc -c -gstabs test.c`

**Major Command-line Options**

-S	Output assembly code (.s)
-c	Output relocatable object file (.o)
-E	Execute C preprocessor only
-B<path>	Specify compiler search path
-I<path>	Specify include file directory
-fno-builtin	Disable built-in functions
-D<macro>[=<string>]	Define macro name
-O	Optimization
-gstabs	Add debugging information with relative path to source files
-mpointer16	Generate code for 16-bit (64KB) data space
-mshort-offset	Generate code for 20-bit (1MB) space
-xassembler-with-cpp	Invoke C preprocessor
-Wa, <option>	Specify assembler option

**Outline**

This tool assembles assembly source files output by the C compiler and converts the mnemonics of the source files into object codes (machine language) of the S1C17. The **as.exe** allows the user to invoke the assembler through **xgcc.exe**, this makes it possible to include preprocessor directives into assembly source files. The results are output in an object file that can be linked or added to a library.

**Flowchart****Start-up Command**

```
as <options> <filename>
```

<filename> Assembly source file name

Example: as -otest.o -adh1 test.s

**Major Command-line Options**

-o<filename>	Specify output file name
-a[<suboption>]	Output assembly list file Example: -adh1 (high-level assembly listing without debugging directives)
--gstabs	Add debugging information with relative path to source files
-mpointer16	Specify 16-bit pointer mode
-mcl7_ext <dumpfile> <mapfile>	Optimize extended instructions

**Major Preprocessor Pseudo-instructions**

#include	Insertion of file
#define	Definition of character strings and numbers
#macro - #endm	Definition of macros
#if - #else - #endif	Conditional assembly (Can be used when the -c -xassembler-with-cpp option of xgcc is specified.)

**Major Assembler Pseudo-instructions**

.text	Declare .text section
.section .data	Declare .data section
.section .rodata	Declare .rodata section
.section .bss	Declare .bss section
.long <data>	Define 4-byte data
.short <data>	Define 2-byte data
.byte <data>	Define 1-byte data
.ascii <string>	Define ASCII character strings
.space <length>	Define blank area (0x0)
.zero <length>	Define blank area (0x0)
.align <value>	Alignment to specify boundary address
.global <symbol>	Global declaration of symbol
.set <symbol>, <address>	Define symbol with absolute address

## Error/Warning Messages

## Error messages

Unrecognized opcode: 'XXXXX'	The operation code XXXXX is undefined.
junk at end of line: 'XXXXX'	A format error of the operand.
XXXXXX: invalid register name	The specified register cannot be used.
operand out of range (XXXXXX: XXX not between AAA and BBB)	The value specified in the operand is out of the effective range.
There are too many characters of one line in assembler source file.	The number of characters (except for a new line character) in an assembler source line has exceeded 2,047 characters.
Cannot allocate memory.	Memory allocation by <code>malloc()</code> has failed.
Cannot specify plurality source files.	More than one source file name is specified in the command line.
Cannot find the dump file.	A dump file name is not specified even though the <code>-mc17_ext</code> option is specified. Or the specified dump file does not exist.
Cannot find the map file.	A map file name is not specified even though the <code>-mc17_ext</code> option is specified. Or the specified map file does not exist.
The format of the dump file is invalid.	The contents in the dump file specified with the <code>-mc17_ext</code> option are invalid.
The format of the map file is invalid.	The contents of the map file specified with the <code>-mc17_ext</code> option are invalid.
Cannot close the map file.	The map file specified with the <code>-mc17_ext</code> option cannot be closed after it has been read.
There are too many characters of one line in dump file.	The number of characters (except for a new line character) in a line of the dump file specified with the <code>-mc17_ext</code> option has exceeded 2,047 characters.
There are too many characters of one line in map file.	The number of characters (except for a new line character) in a line of the map file specified with the <code>-mc17_ext</code> option has exceeded 2,047 characters.

## Warning messages

Unrecognized .section attribute: want a, w, x	The section attribute is not a, w or x.
Bignum truncated to AAA bytes	The constant declared (e.g. <code>.long, .int</code> ) exceeds the maximum size. It has been corrected to AAA-byte size.
Value XXXX truncated to AAA	The constant declared exceeds the maximum value AAA. It has been corrected to AAA.

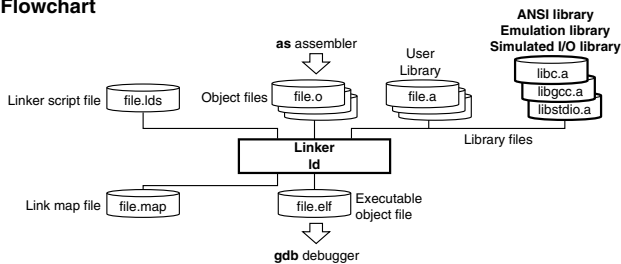
## Linker ld (1)

## Development Tools

### Outline

Defines the memory locations of object codes created by the C compiler and assembler, and creates executable object codes. This tool puts together multiple objects and library files into one file.

### Flowchart



### Start-up Command

```
ld <options> <filename>
```

<filename> Object and library files to be linked

```
Example: ld -o sample.elf boot.o sample.o ..\lib\24bit\libc.a
        ..\lib\24bit\libgcc.a ..\lib\24bit\libc.a
```

### Major Command-line Options

-o <filename>	Specify output file name
-T <filename>	Read linker script file
-M	Link map stdout output
-Map <filename>	Link map file output
-N	Disable data segment alignment check

### Error Messages

The offset value of a symbol is over 16bit.	The address of the symbol exceeds the 16-bit address space.
The offset value of a symbol is over 24bit.	The address of the symbol exceeds the 24-bit address space.
Input object file use both 16bit and 24bit address mode.	The object files to be linked contain both files created in 24-bit pointer mode and 16-bit pointer mode.
section XXX is not within 16bit address.	The address of the XXX section exceeds the 16-bit address space.
section XXX is not within 24bit address.	The address of the XXX section exceeds the 24-bit address space.

**Default linker script file generated by the IDE**

```

/* Linker Script file generated by Gnu17 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);

SECTIONS
{
  /* location counter */
  . = 0x0;

  /* section information */
  .bss 0x000000 :
  {
    _START_bss = . ;
    boot.o(.bss)
    main.o(.bss)
    libc.a(.bss)
  } _END_bss = . ;

  .data _END_bss : AT( _END_rodاتا )
  {
    _START_data = . ;
    files(.data)
  } _END_data = . ;

```

```

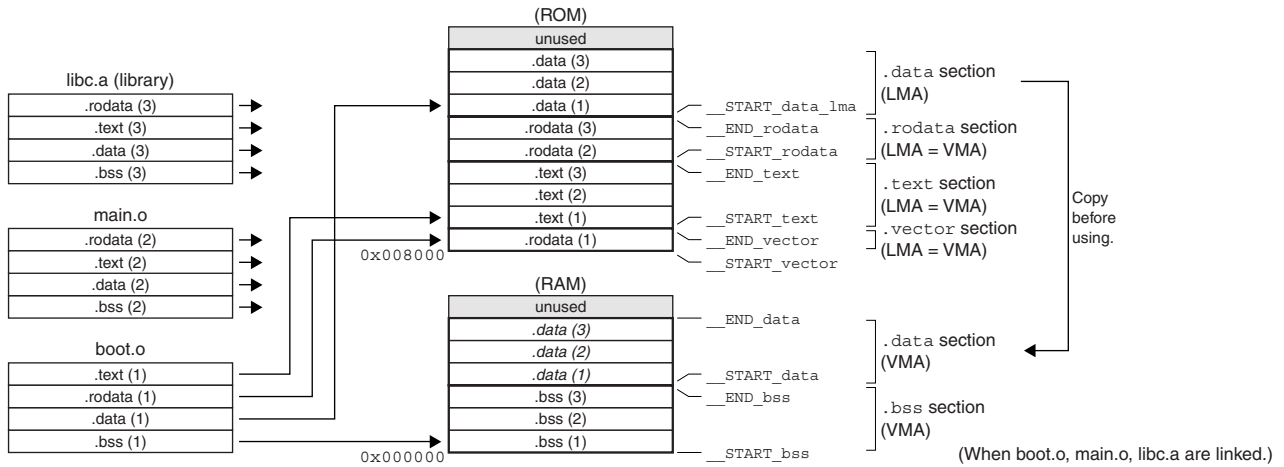
.vector 0x008000 :
{
  _START_vector = . ;
  boot.o(.rodاتا)
} _END_vector = . ;

.text _END_vector :
{
  _START_text = . ;
  files(.text)
} _END_text = . ;

.rodاتا _END_text :
{
  _START_rodاتا = . ;
  main.o(.rodاتا)
  libc.a(.rodاتا)
} _END_rodاتا = . ;

/* load address symbols */
_START_data_lma = LOADADDR( .data );
_END_data_lma = _START_data_lma + SIZEOF( .data );

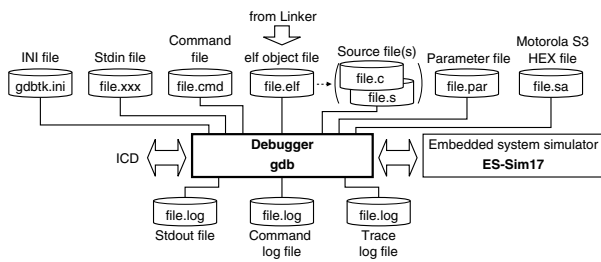
```



**Outline**

The **gdb** serves to perform source-level debugging by controlling an ICD. It also comes with a simulating function that allows you to perform debugging on a personal computer. **gdb.exe** supports Windows GUI. Commands that are used frequently, such as break and step, are registered on the tool bar, minimizing the necessary keyboard operations. Moreover, various data can be displayed in multi windows, with resultant increased efficiency in the debugging tasks.

**Flowchart**



**Start-up Command**

**gdb** <options>

Example: `gdb -x sample.cmd --cd=/cygdrive/c/EPSON/gnu17/sample`

**Command-line Options**

<code>--command=&lt;filename&gt;</code>	Specifies a command file
<code>-x &lt;filename&gt;</code>	Specifies a command file
<code>--c17_cmw=&lt;seconds&gt;</code>	Specifies the command execution intervals for command files
<code>--cd=&lt;path&gt;</code>	Changes current directory
<code>--directory=&lt;path&gt;</code>	Changes source file directory
<code>--c17_euc</code>	Displays the EUC code
<code>--c17_double_starting</code>	Enables double starting

Windows

Forced break button

Program execution buttons

Combo box to display and search source file names

User defined button

Reset button

Combo box to display and search function names

Window open buttons

Pull-down list for selecting a display mode (SOURCE/ASSEMBLY/MIXED/SRC+ASM)

Current PC address and source line number

Connect mode and CPU type

Text box for searching strings

Text box for specifying display start address

[Source] window

This is the main window of the debugger. The menus and the tool bar are generally used in the debugger. This window allows execution of some debugging functions, such as setting/clearing software PC break points, as well as displaying programs.

[Trace] window

Used to display trace data.

[Simulated I/O] window

Used to input and display data for simulated I/O function.

[Console] window

Used to enter debug commands and display the execution results.

[Registers] window

Used to display and edit the register values.

[Watch Expressions] window

Used to monitor global symbols and register values.

[Local Variables] window

Used to monitor local variables.

[Breakpoints] window

Used to manage software PC breakpoints.

[Memory] window

Used to display and edit memory data.

**Menus**

**[Source] window**

<b>File</b>	
Open...	Ctrl+O
Close	Ctrl+W
Source...	
Page Setup...	
Print Source...	
Exit	

<b>User</b>	
User Command	

<b>Reset</b>	
Reset	

<b>View</b>	
Registers	Ctrl+R
Memory	Ctrl+M
Watch Expressions	Ctrl+T
Local Variables	Ctrl+L
Breakpoints	Ctrl+B
Console	Ctrl+N
Sim/I/O	Ctrl+I
Trace	Ctrl+A

<b>Control</b>	
Step	S
Next	N
Finish	F
Continue	C
Step Asm Inst	S
Next Asm Inst	N

**[File] menu**

**Open...** Executes the `file` command after selecting an elf object file.  
**Close** Closes the elf object file currently opened.  
**Source...** Executes the `source` command after selecting a command file.  
**Page Setup...** Sets paper size and margins.  
**Print Source...** Prints the source currently displayed.  
**Exit** Terminates the debugger.

**[User] menu**

**User Command** Executes the command file `userdefine.gdb`.

**[Reset] menu**

**Reset** Executes the command file `reset.gdb` (`c17 rst` command).

**[View] menu**

**Registers** Opens the [Registers] window.  
**Memory** Opens the [Memory] window.  
**Watch Expressions** Opens the [Watch Expressions] window.  
**Local Variables** Opens the [Local Variables] window.  
**Breakpoints** Opens the [Breakpoints] window.  
**Console** Opens the [Console] window.  
**Sim/I/O** Opens the [Simulated I/O] window.  
**Trace** Opens the [Trace] window.

**[Control] menu**

**Step** Issues the `step` command to execute one source line of the target program at the current PC address.  
**Next** Issues the `next` command to execute one source line of the target program at the current PC address. Functions and subroutines are executed as 1 step.  
**Finish** Issues the `finish` command to execute the target program from the current PC address. The program stops when it exits from the currently executed function.  
**Continue** Issues the `continue` command to execute the program continuously from the current PC address.  
**Step Asm Inst** Issues the `stepi` command to execute one instruction step of the target program at the current PC address.  
**Next Asm Inst** Issues the `nexti` command to execute one instruction step of the target program at the current PC address. Subroutines are executed as 1 step.

**Menus**

**[Source] window**

<b>Preferences</b>	
Global...	
Source...	

<b>Help</b>	
About GDB...	

**[Registers] window**

<b>Register</b>	
Edit	
Format	▶
Remove from Display	
Add to Watch	
Display All Registers	

**[Memory] window**

<b>Addresses</b>	
Update Now	Ctrl+U
Preferences...	

**[Breakpoints] window**

<b>breakpoint</b>	
Enabled	
Disabled	
Remove	

**[Preferences] menu**

**Global...** Displays the [Global Preferences] dialog box used to select the fonts.  
**Source...** Displays the [Source Preferences] dialog box for selecting source display colors and tab width.

**[Help] menu**

**About GDB...** Displays the version of the debugger.

**[Register] menu**

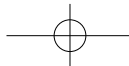
**Edit** Sets the register selected in the window into the edit mode allowing change to the register value.  
**Format** Selects a display format for the register selected in the window.  
**Remove from Display** Removes the selected register from the window.  
**Add to Watch** Adds the register selected in the window to the watch list of the [Watch Expressions] window.  
**Display All Registers** Displays the all register values. Can be used to redisplay the registers removed from the window by the [Remove from Display] command.

**[Addresses] menu**

**Update Now** Update the display contents.  
**Preferences...** Displays the [Memory Preferences] dialog box to select a data format and other conditions for displaying memory contents.

**[Breakpoint] menu**

**Enabled** Enables the breakpoint selected in the window.  
**Disabled** Disables the breakpoint selected in the window.  
**Remove** Clears the breakpoint selected in the window.





**Menus**

**[Breakpoints] window**



**[Global] menu**

**Disable All** Disables all the PC breakpoints that have been set.  
**Enable All** Enables all the PC breakpoints that have been set.  
**Remove All** Clears all the PC breakpoints that have been set.

**[Watch Expressions] window**



**[Watch] menu**

**Edit** Edits the symbol selected in the window.  
**Format** Selects a display format for the symbol selected in the window.  
**Remove** Removes the symbol selected in the window from the watch list.

**[Local Variables] window**



**[Variable] menu**

**Edit** Edits the variable selected in the window.  
**Format** Selects a display format for the variable selected in the window.

**List of Shortcut Keys Allowed in the gdb Main Screen**

S	Executes the <code>step</code> command in Source or SRC+ASM display mode (otherwise, <code>steppi</code> ).
N	Executes the <code>next</code> command in Source or SRC+ASM display mode (otherwise, <code>nexti</code> ).
F	Execute the <code>finish</code> command.
C	Execute the <code>continue</code> command.
Ctrl + O	Opens the target file.
Ctrl + W	Closes the elf file being debugged.
Ctrl + R	Opens the [Registers] window.
Ctrl + M	Opens the [Memory] window.
Ctrl + T	Opens the [Watch Expressions] window.
Ctrl + L	Opens the [Local Variables] window.
Ctrl + B	Opens the [Breakpoints] window.
Ctrl + N	Opens the [Console] window.
Ctrl + I	Opens the [Simulated I/O] window.

## Debug Commands

## Memory operation

<b>c17 fb</b> <i>addr1 addr2 data</i>	Fill memory area (8 bits)	ICD Mini/SIM
<b>c17 fh</b> <i>addr1 addr2 data</i>	Fill memory area (16 bits)	ICD Mini/SIM
<b>c17 fw</b> <i>addr1 addr2 data</i>	Fill memory area (32 bits)	ICD Mini/SIM
<b>x</b> <i>f[length]b [addr]</i>	Dump memory data (8 bits)	ICD Mini/SIM
<b>x</b> <i>f[length]h [addr]</i>	Dump memory data (16 bits)	ICD Mini/SIM
<b>x</b> <i>f[length]w [addr]</i>	Dump memory data (32 bits)	ICD Mini/SIM
<b>set (char)</b> <i>addr=data</i>	Set memory data (8 bits)	ICD Mini/SIM
<b>set (short)</b> <i>addr=data</i>	Set memory data (16 bits)	ICD Mini/SIM
<b>set (long)</b> <i>addr=data</i>	Set memory data (32 bits)	ICD Mini/SIM
<b>c17 mvb</b> <i>addr1 addr2 addr3</i>	Copy memory area (8 bits)	ICD Mini/SIM
<b>c17 mvh</b> <i>addr1 addr2 addr3</i>	Copy memory area (16 bits)	ICD Mini/SIM
<b>c17 mvw</b> <i>addr1 addr2 addr3</i>	Copy memory area (32 bits)	ICD Mini/SIM
<b>c17 df</b> <i>addr1 addr2 type file [append]</i>	Save memory data to file	ICD Mini/SIM
<b>c17 readmd</b> <i>mode</i>	Memory read mode	ICD Mini

## Register operation

<b>info reg</b> [ <i>register</i> ]	Display register data	ICD Mini/SIM
<b>set \$</b> <i>register</i> = <i>data</i>	Set register data	ICD Mini/SIM

## Program execution

<b>continue</b> [ <i>ignore</i> ]	Execute program successively	ICD Mini/SIM
<b>until</b> <i>addr</i>	Execute program successively with temporary break	ICD Mini/SIM
<b>step</b> [ <i>count</i> ]	Execute source lines	ICD Mini/SIM
<b>stepi</b> [ <i>count</i> ]	Execute instruction steps	ICD Mini/SIM
<b>next</b> [ <i>count</i> ]	Execute source lines with function skip	ICD Mini/SIM
<b>nexti</b> [ <i>count</i> ]	Execute instruction steps with subroutine skip	ICD Mini/SIM
<b>finish</b>	Exit from function/subroutine	ICD Mini/SIM
<b>c17 callmd</b> <i>mode [file]</i>	Set user-function call mode	ICD Mini/SIM
<b>c17 call</b> <i>func [arg1... [arg3]]</i>	Call user function	ICD Mini/SIM

## CPU reset

<b>c17 rst</b>	Reset CPU (execute reset.gdb)	ICD Mini/SIM
<b>c17 rstt</b>	Reset target	ICD Mini

## Interrupt

<b>c17 int</b> [ <i>intNo. level</i> ]	Generate interrupt	SIM
<b>c17 intclear</b> [ <i>intNo.</i> ]	Clear interrupt	SIM
<b>c17 int_load</b> [ <i>file</i> ]	Load interrupt event file	SIM

## Break

<b>break</b> [ <i>addr</i> ]	Set software PC breakpoint	ICD Mini/SIM
<b>tbreak</b> [ <i>addr</i> ]	Set temporary software PC breakpoint	ICD Mini/SIM
<b>hbreak</b> [ <i>addr</i> ]	Set hardware PC breakpoint	ICD Mini/SIM
<b>thbreak</b> [ <i>addr</i> ]	Set temporary hardware PC breakpoint	ICD Mini/SIM
<b>delete</b> [ <i>breakNo.</i> ]	Clear breakpoint by break number	ICD Mini/SIM
<b>clear</b> [ <i>addr</i> ]	Clear breakpoint by location	ICD Mini/SIM
<b>enable</b> [ <i>breakNo.</i> ]	Enable breakpoint	ICD Mini/SIM
<b>disable</b> [ <i>breakNo.</i> ]	Disable breakpoint	ICD Mini/SIM
<b>ignore</b> <i>breakNo. count</i>	Disable breakpoint with ignore count	ICD Mini/SIM
<b>info breakpoints</b>	Display breakpoint list	ICD Mini/SIM
<b>c17 timebrk</b> <i>timer</i>	Set lapse of time break	ICD Mini

## Symbol information

<b>info locals</b>	Display local symbol information	ICD Mini/SIM
<b>info var</b>	Display global symbol information	ICD Mini/SIM
<b>print</b> <i>symbol[=value]</i>	Change symbol values	ICD Mini/SIM

## File

<b>file</b> <i>file</i>	Load debug information	ICD Mini/SIM
<b>load</b> [ <i>file</i> ]	Load program	ICD Mini/SIM
<b>c17 loadmd</b> <i>mode</i>	Set program load mode	ICD Mini

## Map information

<b>c17 rpf</b> <i>file</i>	Set map information	SIM
<b>c17 map</b>	Display map information	SIM

## Flash memory

<b>c17 fls</b> <i>addr1 addr2 erase write</i>	Set up flash memory	ICD Mini
<b>c17 fle</b> <i>control blk1 blk2 [timer]</i>	Erase flash memory	ICD Mini

## Trace

<b>c17 tm on/off</b> <i>mode [file]</i>	Set trace mode	SIM
---	----------------	-----

## Simulated I/O

<b>c17 stdin</b> <i>1/2 break buffer [file]</i>	Sets the simulated input condition.	ICD Mini/SIM
<b>c17 stdout</b> <i>1/2 break buffer [file]</i>	Sets the simulated output condition.	ICD Mini/SIM

## Flash writer (for S5U1C17001H)

<b>c17 fwe</b> <i>0/1</i>	Erases program/data.	ICD Mini
<b>c17 fwlp</b> <i>file erase write [comment]</i>	Loads program.	ICD Mini
<b>c17 fwdl</b> <i>file blk1 blk2 par [comment]</i>	Loads data.	ICD Mini
<b>c17 fwdc</b> <i>addr sz blk1 blk2 par [comm]</i>	Copies target memory.	ICD Mini
<b>c17 fwd</b>	Displays flash writer information.	ICD Mini

## Debug Commands

## Others

<b>c17 log</b> [ <i>file</i> ]	Logging	ICD Mini/SIM
<b>source</b> <i>file</i>	Execute command file	ICD Mini/SIM
<b>c17 clockmd</b> <i>mode</i>	Set execution counter mode	ICD Mini/SIM
<b>c17 clock</b>	Display execution counter	ICD Mini/SIM
<b>target</b> <i>type</i>	Connect target	ICD Mini/SIM
<b>detach</b>	Disconnect target	ICD Mini/SIM
<b>pwd</b>	Display current directory	ICD Mini/SIM
<b>cd</b> <i>directory</i>	Change current directory	ICD Mini/SIM
<b>c17 firmupdate</b> <i>file</i>	Update ICD firmware	ICD Mini
<b>c17 ttbr</b> <i>addr</i>	Set TTBR	SIM
<b>c17 help</b> [ <i>command/groupNo.</i> ]	Help	ICD Mini/SIM
<b>quit</b>	Terminate debugger	ICD Mini/SIM

## Status and Error Messages

## Status messages

Breakpoint #, <i>function at file:line</i>	Made to break at the set breakpoint.
Break by accessing no map.	Made to break by the accessing of an unmapped area in simulator mode.
Break by writing ROM area.	Made to break by the accessing of a read-only area in simulator mode.
Break by stack overflow.	Made to break by a stack overflow that occurred in simulator mode.
Illegal instruction.	Made to break by the execution of an illegal instruction in simulator mode.
Illegal delayed instruction.	Made to break by the execution of an illegal delayed instruction in simulator mode.
Break by key break.	Forcibly made to break using the [Stop] button. (Simulator mode)
Break by key break. Program received signal SIGINT, Interrupt.	Forcibly made to break using the [Stop] button. (ICD Mini mode)

## Error messages

... gdb : unrecognized option 'option'	An illegal startup option is specified.
A setup of a serial port was not completed. Address is 24bit over.	An ICD mode not supported in <b>gdb</b> is specified. The specified address is out of the 24-bit range. The maximum S1C17 address size is 24 bits (0xffff).
Address(0x#) is ext or delayed instruction.	The specified address cannot be set, as it is for the <b>ext</b> or delayed instruction.
C17 command error, command is not supported at present mode.	The input command cannot be executed in the current connect mode (ICD Mini or simulator mode).
C17 command error, command is too long.	The input command is too long, as it exceeds 256 characters.
C17 command error, invalid command.	The command is invalid; it contains an error.
C17 command error, invalid parameter.	The command is specified with an invalid parameter.
C17 command error, number of parameter.	The number of parameters in the command is incorrect.
C17 command error, start address > end address.	The number used as the start address specified in the command is greater than that used as the end address.
C17 command error, start cycle < end cycle.	The number used for the end cycle specified in the command is greater than that used for the start cycle.
Cannot access memory at address #	Address # cannot be accessed.
Cannot allocate memory.	A memory area of the size specified by a parameter cannot be allocated.
Cannot clear hard pc break(0x#).	The specified hardware PC break address is nonexistent.
Cannot clear soft pc break(0x#).	The specified software PC break address is nonexistent.

## Status and Error Messages

## Error messages

Cannot display clock counter. Now Timer break mode is on. Please timer break mode off.	The execution counter value cannot be displayed when the lapse of time break is enabled. Disable the lapse of time break before execution times can be measured.
Cannot display clock counter. Time measurement should use continue or until command.	The execution counter value cannot be displayed if the <code>continue</code> or <code>until</code> command has not been executed.
Cannot measure clock timer.	The program execution time cannot be measured as it is too short.
Cannot open file( <i>file</i> ).	The system cannot open the file.
Cannot open ICD17 usb driver.	The system failed to open the USB driver.
Cannot set hard pc break any more.	The number of hardware PC breakpoints set exceeds the limit (1).
Cannot set soft pc break any more.	The number of software PC breakpoints set exceeds the limit (200).
Cannot set timer. Timer Conditions: 1<=Timer<=300000	Cannot set a lapse of time break as the specified time exceeds the valid range.
Cannot write file.	The system cannot write to the file.
Clock timer overflow.	The clock measurement timer has overflowed.
Communication error(bcc).	A BCC error was found in the messages received from the ICD.
Communication system error(#).	The connection was unexpectedly terminated during communication with the ICD.
Copy end address max(0x#) overflow.	The end address of the source to be copied exceeds the upper limit (0xfffff).
Copy start address max(0x#) overflow.	The start address of the source to be copied exceeds the upper limit (0xfffff).
CPU is running.	Cannot accept a command while the CPU is running.
Erase entry address max(0x#) overflow.	The flash erase routine address exceeds the upper limit (0xfffff).
Flash memory end address max(0x#) overflow.	The flash memory end address exceeds the upper limit (0xfffff).
Flash memory start address max(0x#) overflow.	The flash memory start address exceeds the upper limit (0xfffff).
ICD17 is busy(#).	The ICD side is busy.
Initialization error of ICD17.	The system failed to initialize the target.
Invalid ID error(0x#).	The <b>gdb</b> has transmit an invalid ID number. (Internal error)
Invalid parameter file(#. <i>file</i> ).	The parameter file contains an error.
Invalid parameter file, start address > end address(#. <i>file</i> ).	The address range set in the parameter file is invalid, as the number used as the start address is greater than that used as the end address.

## Error messages

It has not connected with a target. It is not c17 architecture ELF file.	The ICD cannot be connected to the target. The file specified with the <code>file</code> command is not an elf format file supported in SSU1C17001C.
Load end address max(0x#) overflow.	The end address of the program to be written to flash memory exceeds the upper limit (0xfffff).
Load max address(0x#) overflow.	The addresses to be loaded exceed the maximum value.
Load motorola file format error.( <i>file</i> )	The specified Motorola file contains a format error.
Load size limit(0x#) overflow.	The size of the specified file exceeds the upper limit. <ul style="list-style-type: none"> <li>Flash write/erase program 8K bytes - 1 byte (0x1fff)</li> <li>Write data for flash memory 3M bytes - 1 byte (0x2ffff)</li> <li>Firmware 8M bytes - 1 byte (0x7ffff)</li> </ul>
Load start address max(0x#) overflow.	The start address of the program to be written to flash memory exceeds the upper limit (0xfffff).
Receiving message is inaccurate.	A message exceeding the maximum size was received during communication with the ICD.
Specification is required in the device for connecting.	There is an insufficient number of device names from which to select the ICD in the <code>target</code> command.
Target down.	A communication error has occurred between the ICD and the target.
There is no argument given to this command.	The system failed to disconnect the target.
Timeout error #(ICD17 -> host).	In communication with the ICD, a receive operation timed out during the wait for data from the ICD.
Too much event(#).	The number of events specified with the <code>c17 int_load</code> (event file read) command exceeds the upper limit (256).
Transmitting failure(#).	NAK was received from the ICD during communication with the ICD.
USB communication error(host->ICD17).	The system failed in USB transmission to the ICD.
USB communication error(ICD17->host).	The system failed in USB receiving from ICD.
Write entry address max(0x#) overflow.	The address of the flash write routine exceeds the upper limit (0xfffff).

## Floating-point Calculation Functions

## Double-type operation

__addf3	Addition	$x \leftarrow a + b$
__subdf3	Subtraction	$x \leftarrow a - b$
__muldf3	Multiplication	$x \leftarrow a * b$
__divdf3	Division	$x \leftarrow a / b$
__negdf2	Sign change	$x \leftarrow -a$

## Float-type operation

__addsf3	Addition	$x \leftarrow a + b$
__subsf3	Subtraction	$x \leftarrow a - b$
__mulsf3	Multiplication	$x \leftarrow a * b$
__divsf3	Division	$x \leftarrow a / b$
__negsf2	Sign change	$x \leftarrow -a$

## Type conversion

__fixunsdfsi	double → unsigned int	$x \leftarrow a$
__fixdfsi	double → int	$x \leftarrow a$
__floatsidf	int → double	$x \leftarrow a$
__fixunssfsi	float → unsigned int	$x \leftarrow a$
__fixsfsi	float → int	$x \leftarrow a$
__floatsfsf	int → float	$x \leftarrow a$
__truncdfsf2	double → float	$x \leftarrow a$
__extendsfdf2	float → double	$x \leftarrow a$

## Comparison

__fcmpd	double type	Changes %psr by a - b
__**df2	double type	Changes %psr and x by a - b **=eq, ne, gt, ge, lt, le (x = 1 if true, 0 if false)
__fcmps	float type	Changes %psr by a - %13
__**sf2	float type	Changes %psr and x by a - %13 **=eq, ne, gt, ge, lt, le (x = 1 if true, 0 if false)

## Floating-point Data Format

## Double-type data format

63	62	52	51	0
S	Exponent part		Fixed-point part	

## Double-type effective range

+0:	0.0e+0 0x00000000 00000000
-0:	-0.0e+0 0x80000000 00000000
Maximum normalized number:	1.79769e+308 0x7feffff ffffffff
Minimum normalized number:	2.22507e-308 0x00100000 00000000
Maximum unnormalized number:	2.22507e-308 0x000ffff ffffffff
Minimum unnormalized number:	4.94065e-324 0x00000000 00000001
Infinity:	0x7ff00000 00000000
-Infinity:	0xff000000 00000000

## Float-type data format

31	30	23	22	0
S	Exponent part		Fixed-point part	

## Float-type effective range

+0:	0.0e+0f 0x00000000
-0:	-0.0e+0f 0x80000000
Maximum normalized number:	3.40282e+38f 0x7fffffff
Minimum normalized number:	1.17549e-38f 0x00800000
Maximum unnormalized number:	1.17549e-38f 0x007ffff
Minimum unnormalized number:	1.40129e-45f 0x00000001
Infinity:	0x7f800000
-Infinity:	0xff800000

**Integral Calculation Functions****Integral calculation**

<code>__divsi3</code>	Signed 32-bit integral division	$x \leftarrow a / b$
<code>__modsi3</code>	Signed 32-bit remainder calculation	$x \leftarrow a \% b$
<code>__udivsi3</code>	Unsigned 32-bit integral division	$x \leftarrow a / b$
<code>__umodsi3</code>	Unsigned 32-bit remainder calculation	$x \leftarrow a \% b$
<code>__muls3</code>	32-bit multiplication	$x \leftarrow a * b$
<code>__divhi3</code>	Signed 16-bit integral division	$x \leftarrow a / b$
<code>__modhi3</code>	Signed 16-bit remainder calculation	$x \leftarrow a \% b$
<code>__udivhi3</code>	Unsigned 16-bit integral division	$x \leftarrow a / b$
<code>__umodhi3</code>	Unsigned 16-bit remainder calculation	$x \leftarrow a \% b$
<code>__mulhi3</code>	16-bit multiplication	$x \leftarrow a / b$

**Integral shift**

<code>__ashisi3</code>	32-bit arithmetical shift to left	$x \leftarrow a \ll b \text{ bits}$
<code>__ashrsi3</code>	32-bit arithmetical shift to right	$x \leftarrow a \gg b \text{ bits}$
<code>__lshrsi3</code>	32-bit logical shift to right	$x \leftarrow a \gg b \text{ bits}$
<code>__ashlhi3</code>	16-bit arithmetical shift to left	$x \leftarrow a \ll b \text{ bits}$
<code>__ashrhi3</code>	16-bit arithmetical shift to right	$x \leftarrow a \gg b \text{ bits}$
<code>__lshrhi3</code>	16-bit logical shift to right	$x \leftarrow a \gg b \text{ bits}$

**long long Type Calculation Functions****long long type calculation**

<code>__adddi3</code>	Signed 64-bit addition	$x \leftarrow a + b$
<code>__subdi3</code>	Signed 64-bit subtraction	$x \leftarrow a - b$
<code>__muldi3</code>	Signed 64-bit multiplication	$x \leftarrow a * b$
<code>__divdi3</code>	Signed 64-bit division	$x \leftarrow a / b$
<code>__udivdi3</code>	Unsigned 64-bit division	$x \leftarrow a / b$
<code>__moddi3</code>	Signed 64-bit remainder calculation	$x \leftarrow a \% b$
<code>__umoddi3</code>	Unsigned 64-bit remainder calculation	$x \leftarrow a \% b$
<code>__negdi2</code>	Sign inversion	$x \leftarrow -a$

**long long type shift**

<code>__lshrdi3</code>	64-bit logical shift to right	$x \leftarrow a \gg b \text{ bits}$
<code>__ashldi3</code>	64-bit arithmetical shift to left	$x \leftarrow a \ll b \text{ bits}$
<code>__ashrdi3</code>	64-bit arithmetical shift to right	$x \leftarrow a \gg b \text{ bits}$

**Type conversion**

<code>__fixunsdfdi</code>	double → unsigned long long	$x \leftarrow a$
<code>__fixdfdi</code>	double → long long	$x \leftarrow a$
<code>__floatdidf</code>	long long → double	$x \leftarrow a$
<code>__fixunssfdi</code>	float → unsigned long long	$x \leftarrow a$
<code>__fixsfdi</code>	float → long long	$x \leftarrow a$
<code>__floatdisf</code>	long long → float	$x \leftarrow a$

**long long type comparison**

<code>__cmpdi2</code>	Comparison (long long)	$x \leftarrow 2   1   0$
<code>__ucmpdi2</code>	Comparison (unsigned long long)	$x \leftarrow 2   1   0$

**Other**

<code>__ffsdi2</code>	Bit scan	$x \leftarrow 64 \text{ to } 0$
-----------------------	----------	---------------------------------

**Input/Output Functions** (header file: stdio.h)

fopen()	FILE *fopen(char *filename, char *mode);	(dummy)*1
freopen()	FILE *freopen(char *filename, char *mode, FILE *stream);	(dummy)*1
fclose()	int fclose(FILE *stream);	(dummy)
fflush()	int fflush(FILE *stream);	(dummy)
fseek()	int fseek(FILE *stream, long int offset, int origin);	(dummy)*1
ftell()	long int ftell(FILE *stream);	(dummy)
rewind()	void rewind(FILE *stream);	(dummy)
fgetpos()	int fgetpos(FILE *stream, fpos_t *ptr);	(dummy)
fsetpos()	int fsetpos(FILE *stream, fpos_t *ptr);	(dummy)*1
fread()	size_t fread(void *ptr, size_t size, size_t count, FILE *stream);	*1, *2
fwrite()	size_t fwrite(void *ptr, size_t size, size_t count, FILE *stream);	*1, *2
fgetc()	int fgetc(FILE *stream);	*2
getc()	int getc(FILE *stream);	*1, *2
getchar()	int getchar();	*1, *2
ungetc()	int ungetc(int c, FILE *stream);	*1
fgets()	char *fgets(char *s, int n, FILE *stream);	*1, *2
gets()	char *gets(char *s);	*1, *2
fputc()	int fputc(int c, FILE *stream);	*2
putc()	int putc(int c, FILE *stream);	*1, *2
putchar()	int putchar(int c);	*1, *2
fputs()	int fputs(char *s, FILE *stream);	*1, *2
puts()	int puts(char *s);	*1, *2
remove()	int remove(char *filename);	(dummy)*1
rename()	int rename(char *oldname, char *newname);	(dummy)*1
setbuf()	void setbuf(FILE *stream, char *buf);	(dummy)
setvbuf()	int setvbuf(FILE *stream, char *buf, int type, size_t size);	(dummy)
tmpfile()	FILE *tmpfile();	(dummy)*1
tmpnam()	char *tmpnam(char *buf);	(dummy)*1
feof()	int feof(FILE *stream);	(dummy)
ferror()	int ferror(FILE *stream);	(dummy)
clearerr()	void clearerr(FILE *stream);	(dummy)
perror()	void perror(char *s);	*1, *2
fscanf()	int fscanf(FILE *stream, char *format, ...);	*1, *2
scanf()	int scanf(char *format, ...);	*1, *2
sscanf()	int sscanf(char *s, char *format, ...);	*1, *2
fprintf()	int fprintf(FILE *stream, char *format, ...);	*1, *2
printf()	int printf(char *format, ...);	*1, *2
sprintf()	int sprintf(char *s, char *format, ...);	*1, *2
vfprintf()	int vfprintf(FILE *stream, char *format, va_list arg);	*1, *2
vprintf()	int vprintf(char *format, va_list arg);	*1, *2
vsprintf()	int vsprintf(char *s, char *format, va_list arg);	*1, *2

**Utility Functions** (header file: stdlib.h)

malloc()	void *malloc(size_t size);	*1
calloc()	void *calloc(size_t elt_count, size_t elt_size);	*1
free()	void free(void *ptr);	*1
realloc()	void *realloc(void *ptr, size_t size);	*1
system()	int system(char *command);	(dummy)
exit()	void exit(int status);	(dummy)
abort()	void abort();	(dummy)
atexit()	int atexit(void (*func)(void));	(dummy)
getenv()	char *getenv(char *str);	(dummy)
bsearch()	void *bsearch(void *key, void *base, size_t count, size_t size, int (*compare)(void *, void *));	(dummy)
qsort()	void qsort(void *base, size_t count, size_t size, int (*compare)(void *, void *));	(dummy)
abs()	int abs(int x);	(dummy)
labs()	long int labs(long int x);	(dummy)
div()	div_t div(int n, int d);	*1
ldiv()	ldiv_t ldiv(int n, int d);	*1
rand()	int rand();	(dummy)
srand()	void srand(unsigned int seed);	(dummy)
atol()	long int atol(char *str);	(dummy)
atoi()	int atoi(char *str);	*1
atof()	double atof(char *str);	*1
strtod()	double strtod(char *str, char **ptr);	*1
strtol()	long int strtol(char *str, char **ptr, int base);	*1
strtoul()	unsigned long int strtoul(char *str, char **ptr, int base);	*1

**Date and Time Functions** (header file: time.h)

clock()	clock_t clock();	(dummy)
asctime()	char *asctime(struct tm *ts);	(dummy)
ctime()	char *ctime(time_t *timeptr);	(dummy)
difftime()	double difftime(time_t t1, time_t t2);	(dummy)
gmtime()	struct tm *gmtime(time_t *t);	(dummy)
localtime()	struct tm *localtime(time_t *t);	(dummy)
mktime()	time_t mktime(struct tm *tmptr);	(dummy)
time()	time_t time(time_t *t);	*1

**Non-local Branch Functions** (header file: setjmp.h)

setjmp()	int setjmp(jmp_buf *ptr);	(dummy)
longjmp()	void longjmp(jmp_buf *ptr, int status);	(dummy)

\*1 These functions need to declare and initialize the global variables.

\*2 These functions need to define the low-level functions and I/O buffers.

**Mathematical Functions** (header file: math.h, ermo.h, float.h, limits.h)

<code>fabs()</code>	double <code>fabs(double x)</code> ;	*1
<code>ceil()</code>	double <code>ceil(double x)</code> ;	*1
<code>floor()</code>	double <code>floor(double x)</code> ;	*1
<code>fmod()</code>	double <code>fmod(double x, double y)</code> ;	*1
<code>exp()</code>	double <code>exp(double x)</code> ;	*1
<code>log()</code>	double <code>log(double x)</code> ;	*1
<code>log10()</code>	double <code>log10(double x)</code> ;	*1
<code>frexp()</code>	double <code>frexp(double x, int *nptr)</code> ;	*1
<code>ldexp()</code>	double <code>ldexp(double x, int n)</code> ;	*1
<code>modf()</code>	double <code>modf(double x, double *nptr)</code> ;	*1
<code>pow()</code>	double <code>pow(double x, double y)</code> ;	*1
<code>sqrt()</code>	double <code>sqrt(double x)</code> ;	*1
<code>sin()</code>	double <code>sin(double x)</code> ;	*1
<code>cos()</code>	double <code>cos(double x)</code> ;	*1
<code>tan()</code>	double <code>tan(double x)</code> ;	*1
<code>asin()</code>	double <code>asin(double x)</code> ;	*1
<code>acos()</code>	double <code>acos(double x)</code> ;	*1
<code>atan()</code>	double <code>atan(double x)</code> ;	
<code>atan2()</code>	double <code>atan2(double y, double x)</code> ;	*1
<code>sinh()</code>	double <code>sinh(double x)</code> ;	*1
<code>cosh()</code>	double <code>cosh(double x)</code> ;	*1
<code>tanh()</code>	double <code>tanh(double x)</code> ;	

**Character Type Determination/Conversion Functions** (header file: ctype.h)

<code>isalnum()</code>	int <code>isalnum(char c)</code> ;
<code>isalpha()</code>	int <code>isalpha(char c)</code> ;
<code>iscntrl()</code>	int <code>iscntrl(char c)</code> ;
<code>isdigit()</code>	int <code>isdigit(char c)</code> ;
<code>isgraph()</code>	int <code>isgraph(char c)</code> ;
<code>islower()</code>	int <code>islower(char c)</code> ;
<code>isprint()</code>	int <code>isprint(char c)</code> ;
<code>ispunct()</code>	int <code>ispunct(char c)</code> ;
<code>isspace()</code>	int <code>isspace(char c)</code> ;
<code>isupper()</code>	int <code>isupper(char c)</code> ;
<code>isxdigit()</code>	int <code>isxdigit(char c)</code> ;
<code>tolower()</code>	int <code>tolower(char c)</code> ;
<code>toupper()</code>	int <code>toupper(char c)</code> ;

**Variable Argument Macros** (header file: stdarg.h)

<code>va_start()</code>	void <code>va_start(va_list ap, type lastarg)</code> ;
<code>va_arg()</code>	type <code>va_arg(va_list ap, type)</code> ;
<code>va_end()</code>	void <code>va_end(va_list ap)</code> ;

\*1 These functions need to declare and initialize the global variables.

**Character Functions** (header file: string.h)

<code>memchr()</code>	char * <code>memchr(char *s, int c, int n)</code> ;
<code>memcmp()</code>	int <code>memcmp(char *s1, char *s2, int n)</code> ;
<code>memcpy()</code>	char * <code>memcpy(char *s1, char *s2, int n)</code> ;
<code>memmove()</code>	char * <code>memmove(char *s1, char *s2, int n)</code> ;
<code>memset()</code>	char * <code>memset(char *s, int c, int n)</code> ;
<code>strcat()</code>	char * <code>strcat(char *s1, char *s2)</code> ;
<code>strchr()</code>	char * <code>strchr(char *s, int c)</code> ;
<code>strcmp()</code>	int <code>strcmp(char *s1, char *s2)</code> ;
<code>strcpy()</code>	char * <code>strcpy(char *s1, char *s2)</code> ;
<code>strncpy()</code>	size_t * <code>strncpy(char *s1, char *s2, int n)</code> ;
<code>strerror()</code>	char * <code>strerror(int code)</code> ;
<code>strlen()</code>	size_t <code>strlen(char *s)</code> ;
<code>strncat()</code>	size_t <code>strncat(char *s1, char *s2, int n)</code> ;
<code>strncmp()</code>	int <code>strncmp(char *s1, char *s2, int n)</code> ;
<code>strncpy()</code>	char * <code>strncpy(char *s1, char *s2, int n)</code> ;
<code>strpbrk()</code>	char * <code>strpbrk(char *s1, char *s2)</code> ;
<code>strrchr()</code>	char * <code>strrchr(char *s, int c)</code> ;
<code>strspn()</code>	size_t <code>strspn(char *s1, char *s2)</code> ;
<code>strstr()</code>	char * <code>strstr(char *s1, char *s2)</code> ;
<code>strtok()</code>	char * <code>strtok(char *s1, char *s2)</code> ;

**\*1 Declaring and Initializing Global Variables**

<code>FILE _job[FOPEN_MAX+1];</code>	<code>_job[N]._flg=_UGETN; _job[N]._buf=0; _job[N]._fd=N;</code> (N=0: stdin, N=1: stdout, N=2: stderr)
<code>FILE *stdin;</code>	<code>stdin=&amp;_job[0];</code>
<code>FILE *stdout;</code>	<code>stdout=&amp;_job[1];</code>
<code>FILE *stderr;</code>	<code>stderr=&amp;_job[2];</code>
<code>int errno;</code>	<code>errno=0;</code>
<code>unsigned int seed;</code>	<code>seed=1;</code>
<code>time_t gm_sec;</code>	<code>gm_sec=-1;</code>

**\*2 Definition of Lower-level Functions**

<code>read()</code>	int <code>read(int fd, char *buf, int nbytes)</code> ; unsigned char <code>READ_BUF[65]</code> ; (Variable name is arbitrary) unsigned char <code>READ_EOF</code> ;
<code>write()</code>	int <code>write(int fd, char *buf, int nbytes)</code> ; unsigned char <code>WRITE_BUF[65]</code> ; (Variable name is arbitrary)



**Symbols in the Instruction List****Registers/Register Data**

%rd, rd:	A general-purpose register (R0–R7) used as the destination register or its contents
%rs, rs:	A general-purpose register (R0–R7) used as the source register or its contents
%rb, rb:	A general-purpose register (R0–R7) that has stored a base address to be accessed in the register indirect addressing mode or its contents
%sp, sp:	Stack pointer (SP) or its contents
%pc, pc:	Program counter (PC) or its contents

**Memory/Addresses/Memory Data**

[%rb], [%sp]:	Specification for register indirect addressing
[%rb]+, [%sp]+:	Specification for register indirect addressing with post-increment
[%rb]-, [%sp]-:	Specification for register indirect addressing with post-decrement
-%rb, -[%sp]:	Specification for register indirect addressing with pre-decrement
[%sp+immX]:	Specification for register indirect addressing with a displacement
[imm7]:	Specification for a memory address with an immediate data
B[XXX]:	An address specified with XXX, or the byte data stored in the address
W[XXX]:	A 16-bit address specified with XXX, or the word data stored in the address
A[XXX]:	A 32-bit address specified with XXX, or the 24-bit or 32-bit data stored in the address

**Immediate**

immX:	A X-bit unsigned immediate data
signX:	A X-bit signed immediate data

**Symbol/Label**

Symbol:	A symbol that points an address.
Label:	A branch destination label.

**Bit Field**

(X):	Bit X of data.
(X:Y):	A bit field from bit X to bit Y.
{X, Y...}:	Indicates a bit (data) configuration.

**Functions**

<-:	Indicates that the right item is loaded or set to the left item.
+:	Addition
-:	Subtraction
&:	AND
:	OR
^:	XOR
!:	NOT

**Flags**

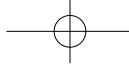
IL:	Interrupt level
IE:	Interrupt enable flag
C:	Carry flag
V:	Overflow flag
Z:	Zero flag
N:	Negative flag
-:	Not changed
<->:	Set (1), reset (0) or not changed
1:	Set (1)
0:	Reset (0)

**D**

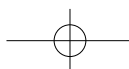
O:	Indicates that the instruction can be used as a delayed instruction.
-:	Indicates that the instruction cannot be used as a delayed instruction.

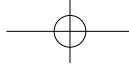
**Notes**

- The instruction list contains the basic instructions in the S1C17 instruction set and the extended instructions (s... and x..., except for xor).
- "*Italic basic instructions*" indicate that the upper compatible extended instructions are provided.

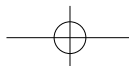
**Instruction List (2)****Assembly Programming**

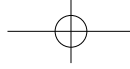
Classification	Mnemonic		Function	Flags						D
	Opcode	Operand		IL	IE	C	V	Z	N	
<b>Signed 8-bit data transfer</b>	ld.b	%rd, %rs	rd(7:0)←rs(7:0), rd(15:8)←rs(7), rd(23:16)←0	-	-	-	-	-	-	○
		%rd, [%rb]	rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0	-	-	-	-	-	-	○
		%rd, [%rb]+	rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0, rb(23:0)←rb(23:0)+1	-	-	-	-	-	-	○
		%rd, [%rb]-	rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0, rb(23:0)←rb(23:0)-1	-	-	-	-	-	-	○
		%rd, -[%rb]	rb(23:0)←rb(23:0)-1, rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0	-	-	-	-	-	-	○
		%rd, [%sp+imm7]	rd(7:0)←B[sp+imm7], rd(15:8)←B[sp+imm7](7), rd(23:16)←0	-	-	-	-	-	-	○
		%rd, [imm7]	rd(7:0)←B[imm7], rd(15:8)←B[imm7](7), rd(23:16)←0	-	-	-	-	-	-	○
		[%rb], %rs	B[rb]←rs(7:0)	-	-	-	-	-	-	○
		[%rb]+, %rs	B[rb]←rs(7:0), rb(23:0)←rb(23:0)+1	-	-	-	-	-	-	○
		[%rb]-, %rs	B[rb]←rs(7:0), rb(23:0)←rb(23:0)-1	-	-	-	-	-	-	○
	-%rs	rb(23:0)←rb(23:0)-1, B[rb]←rs(7:0)	-	-	-	-	-	-	○	
	[%sp+imm7], %rs	B[sp+imm7]←rs(7:0)	-	-	-	-	-	-	○	
	[imm7], %rs	B[imm7]←rs(7:0)	-	-	-	-	-	-	○	
	sld.b	%rd, [%sp+imm20]	%rd←B[%sp+imm20](with sign extension)	-	-	-	-	-	-	-
		%rd, [imm20]	%rd←B[imm20](with sign extension)	-	-	-	-	-	-	-
		[%sp+imm20], %rs	B[%sp+imm20]←%rs(7:0)	-	-	-	-	-	-	-
[imm20], %rs		B[imm20]←%rs(7:0)	-	-	-	-	-	-	-	
xld.b	%rd, [%sp+imm24]	%rd←B[%sp+imm24](with sign extension)	-	-	-	-	-	-	-	
	%rd, [imm24]	%rd←B[imm24](with sign extension)	-	-	-	-	-	-	-	
	[%sp+imm24], %rs	B[%sp+imm24]←%rs(7:0)	-	-	-	-	-	-	-	
	[imm24], %rs	B[imm24]←%rs(7:0)	-	-	-	-	-	-	-	
<b>Unsigned 8-bit data transfer</b>	ld.ub	%rd, %rs	rd(7:0)←rs(7:0), rd(15:8)←0, rd(23:16)←0	-	-	-	-	-	-	○
		%rd, [%rb]	rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0	-	-	-	-	-	-	○
		%rd, [%rb]+	rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0, rb(23:0)←rb(23:0)+1	-	-	-	-	-	-	○
		%rd, [%rb]-	rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0, rb(23:0)←rb(23:0)-1	-	-	-	-	-	-	○
		%rd, -[%rb]	rb(23:0)←rb(23:0)-1, rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0	-	-	-	-	-	-	○
		%rd, [%sp+imm7]	rd(7:0)←B[sp+imm7], rd(15:8)←0, rd(23:16)←0	-	-	-	-	-	-	○
	sld.ub	%rd, [%sp+imm20]	%rd←B[%sp+imm20](with zero extension)	-	-	-	-	-	-	-
		%rd, [imm20]	%rd←B[imm20](with zero extension)	-	-	-	-	-	-	-
	xld.ub	%rd, [%sp+imm24]	%rd←B[%sp+imm24](with zero extension)	-	-	-	-	-	-	-
		%rd, [imm24]	%rd←B[imm24](with zero extension)	-	-	-	-	-	-	-
<b>Remarks</b>										



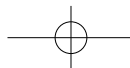


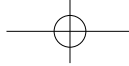
Instruction List (3)			Assembly Programming							
Classification	Mnemonic		Function	Flags					D	
	Opcode	Operand		IL	IE	C	V	Z		N
16-bit data transfer	ld	%rd, %rs	rd(15:0)←rs(15:0), rd(23:16)←0	-	-	-	-	-	-	○
		%rd, sign7	rd(6:0)←sign7(6:0), rd(15:7)←sign7(6), rd(23:16)←0	-	-	-	-	-	-	○
		%rd, [%rb]	rd(15:0)←W[rb], rd(23:16)←0	-	-	-	-	-	-	○
		%rd, [%rb]+	rd(15:0)←W[rb], rd(23:16)←0, rb(23:0)←rb(23:0)+2	-	-	-	-	-	-	○
		%rd, [%rb]-	rd(15:0)←W[rb], rd(23:16)←0, rb(23:0)←rb(23:0)-2	-	-	-	-	-	-	○
		%rd, -[%rb]	rb(23:0)←rb(23:0)-2, rd(15:0)←W[rb], rd(23:16)←0	-	-	-	-	-	-	○
		%rd, [%sp+imm7]	rd(15:0)←W[sp+imm7], rd(23:16)←0	-	-	-	-	-	-	○
		%rd, [imm7]	rd(15:0)←W[imm7], rd(23:16)←0	-	-	-	-	-	-	○
		[%rb], %rs	W[rb]←rs(15:0)	-	-	-	-	-	-	○
		[%rb]+, %rs	W[rb]←rs(15:0), rb(23:0)←rb(23:0)+2	-	-	-	-	-	-	○
		[%rb]-, %rs	W[rb]←rs(15:0), rb(23:0)←rb(23:0)-2	-	-	-	-	-	-	○
		-%rb, %rs	rb(23:0)←rb(23:0)-2, W[rb]←rs(15:0)	-	-	-	-	-	-	○
		[%sp+imm7], %rs	W[sp+imm7]←rs(15:0)	-	-	-	-	-	-	○
		[imm7], %rs	W[imm7]←rs(15:0)	-	-	-	-	-	-	○
	sld	%rd, imm16	%rd←imm16	-	-	-	-	-	-	-
		%rd, symbol±imm16	%rd←symbol±imm16(15:0)	-	-	-	-	-	-	-
		%rd, [%sp+imm20]	%rd←W[%sp+imm20]	-	-	-	-	-	-	-
		%rd, [imm20]	%rd←W[imm20]	-	-	-	-	-	-	-
		[%sp+imm20], %rs	W[%sp+imm20]←%rs(15:0)	-	-	-	-	-	-	-
		[imm20], %rs	W[imm20]←%rs(15:0)	-	-	-	-	-	-	-
	xld	%rd, imm16	%rd←imm16	-	-	-	-	-	-	-
		%rd, symbol±imm16	%rd←symbol±imm16(15:0)	-	-	-	-	-	-	-
		%rd, [%sp+imm24]	%rd←W[%sp+imm24]	-	-	-	-	-	-	-
		%rd, [imm24]	%rd←W[imm24]	-	-	-	-	-	-	-
		[%sp+imm24], %rs	W[%sp+imm24]←%rs(15:0)	-	-	-	-	-	-	-
		[imm24], %rs	W[imm24]←%rs(15:0)	-	-	-	-	-	-	-
32-bit data transfer	ld.a	%rd, %rs	rd(23:0)←rs(23:0)	-	-	-	-	-	-	○
		%rd, imm7	rd(6:0)←imm7(6:0), rd(23:7)←0	-	-	-	-	-	-	○
		%rd, [%rb]	rd(23:0)←A[rb](23:0), ignored←A[rb](31:24)	-	-	-	-	-	-	○
		%rd, [%rb]+	rd(23:0)←A[rb](23:0), ignored←A[rb](31:24), rb(23:0)←rb(23:0)+4	-	-	-	-	-	-	○
		%rd, [%rb]-	rd(23:0)←A[rb](23:0), ignored←A[rb](31:24), rb(23:0)←rb(23:0)-4	-	-	-	-	-	-	○
		%rd, -[%rb]	rb(23:0)←rb(23:0)-4, rd(23:0)←A[rb](23:0), ignored←A[rb](31:24)	-	-	-	-	-	-	○
		%rd, [%sp+imm7]	rd(23:0)←A[sp+imm7](23:0), ignored←A[sp+imm7](31:24)	-	-	-	-	-	-	○
		%rd, [imm7]	rd(23:0)←A[imm7](23:0), ignored←A[imm7](31:24)	-	-	-	-	-	-	○
Remarks										



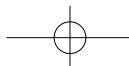
**Instruction List (4)****Assembly Programming**

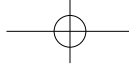
Classification	Opcode	Mnemonic Operand	Function	Flags						D
				IL	IE	C	V	Z	N	
32-bit data transfer	ld.a	[%rb], %rs	A[rb](23:0)←rs(23:0), A[rb](31:24)←0	-	-	-	-	-	-	○
		[%rb]+, %rs	A[rb](23:0)←rs(23:0), A[rb](31:24)←0, rb(23:0)←rb(23:0)+4	-	-	-	-	-	-	○
		[%rb]-, %rs	A[rb](23:0)←rs(23:0), A[rb](31:24)←0, rb(23:0)←rb(23:0)-4	-	-	-	-	-	-	○
		[-%rb], %rs	rb(23:0)←rb(23:0)-4, A[rb](23:0)←rs(23:0), A[rb](31:24)←0	-	-	-	-	-	-	○
		[%sp+imm7], %rs	A[sp+imm7](23:0)←rs(23:0), A[sp+imm7](31:24)←0	-	-	-	-	-	-	○
		[imm7], %rs	A[imm7](23:0)←rs(23:0), A[imm7](31:24)←0	-	-	-	-	-	-	○
		%rd, %sp	rd(23:2)←sp(23:2), rd(1:0)←0	-	-	-	-	-	-	○
		%rd, %pc	rd(23:0)←pc(23:0)+2	-	-	-	-	-	-	○
		%rd, [%sp]	rd(23:0)←A[sp](23:0), ignored←A[sp](31:24)	-	-	-	-	-	-	○
		%rd, [%sp]+	rd(23:0)←A[sp](23:0), ignored←A[sp](31:24), sp(23:0)←sp(23:0)+4	-	-	-	-	-	-	○
		%rd, [%sp]-	rd(23:0)←A[sp](23:0), ignored←A[sp](31:24), sp(23:0)←sp(23:0)-4	-	-	-	-	-	-	○
		%rd, [-%sp]	sp(23:0)←sp(23:0)-4, rd(23:0)←A[sp](23:0), ignored←A[sp](31:24)	-	-	-	-	-	-	○
		[%sp], %rs	A[sp](23:0)←rs(23:0), A[sp](31:24)←0	-	-	-	-	-	-	○
		[%sp]+, %rs	A[sp](23:0)←rs(23:0), A[sp](31:24)←0, sp(23:0)←sp(23:0)+4	-	-	-	-	-	-	○
	[%sp]-, %rs	A[sp](23:0)←rs(23:0), A[sp](31:24)←0, sp(23:0)←sp(23:0)-4	-	-	-	-	-	-	○	
	[-%sp], %rs	sp(23:0)←sp(23:0)-4, A[sp](23:0)←rs(23:0), A[sp](31:24)←0	-	-	-	-	-	-	○	
	%sp, %rs	sp(23:2)←rs(23:2)	-	-	-	-	-	-	○	
	%sp, imm7	sp(6:2)←imm7(6:2), sp(23:7)←0	-	-	-	-	-	-	○	
	sld.a	%rd, imm20	%rd←imm20	-	-	-	-	-	-	-
		%sp, imm20	%sp←imm20	-	-	-	-	-	-	-
		%rd, symbol±imm20	%rd←symbol±imm20(19:0)	-	-	-	-	-	-	-
		%sp, symbol±imm20	%sp←symbol±imm20(19:0)	-	-	-	-	-	-	-
		%rd, [%sp+imm20]	%rd←A[%sp+imm20](23:0), ignored←A[%sp+imm20](31:24)	-	-	-	-	-	-	-
		%rd, [imm20]	%rd←A[imm20](23:0), ignored←A[imm20](31:24)	-	-	-	-	-	-	-
		[%sp+imm20], %rs	A[%sp+imm20](23:0)←%rs(23:0), A[%sp+imm20](31:24)←0	-	-	-	-	-	-	-
		[imm20], %rs	A[imm20](23:0)←%rs(23:0), A[imm20](31:24)←0	-	-	-	-	-	-	-
	xld.a	%rd, imm24	%rd←imm24	-	-	-	-	-	-	-
		%sp, imm24	%sp←imm24	-	-	-	-	-	-	-
		%rd, symbol±imm24	%rd←symbol±imm24(23:0)	-	-	-	-	-	-	-
		%sp, symbol±imm24	%sp←symbol±imm24(23:0)	-	-	-	-	-	-	-
		%rd, [%sp+imm24]	%rd←A[%sp+imm24](23:0), ignored←A[%sp+imm24](31:24)	-	-	-	-	-	-	-
		%rd, [imm24]	%rd←A[imm24](23:0), ignored←A[imm24](31:24)	-	-	-	-	-	-	-
[%sp+imm24], %rs		A[%sp+imm24](23:0)←%rs(23:0), A[imm24](31:24)←0	-	-	-	-	-	-	-	
[imm24], %rs		A[imm24](23:0)←%rs(23:0), A[%sp+imm24](31:24)←0	-	-	-	-	-	-	-	
Remarks										



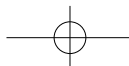
**Instruction List (5)****Assembly Programming**

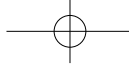
Classification	Mnemonic		Function	Flags						D
	Opcode	Operand		IL	IE	C	V	Z	N	
Arithmetic operation	add	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0), rd(23:16)←0	-	-	↔	↔	↔	↔	○
	add/c	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	-	-	↔	↔	↔	↔	○
	add/nc	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	-	-	↔	↔	↔	↔	○
	add	%rd, imm7	rd(15:0)←rd(15:0)+imm7(with zero extension), rd(23:16)←0	-	-	↔	↔	↔	↔	○
	sadd	%rd, imm16	rd(15:0)←rd(15:0)+imm16, rd(23:16)←0	-	-	↔	↔	↔	↔	-
	xadd	%rd, imm16	rd(15:0)←rd(15:0)+imm16, rd(23:16)←0	-	-	↔	↔	↔	↔	-
	add.a	%rd, %rs	rd(23:0)←rd(23:0)+rs(23:0)	-	-	-	-	-	-	○
	add.a/c	%rd, %rs	rd(23:0)←rd(23:0)+rs(23:0) if C = 1 (nop if C = 0)	-	-	-	-	-	-	○
	add.a/nc	%rd, %rs	rd(23:0)←rd(23:0)+rs(23:0) if C = 0 (nop if C = 1)	-	-	-	-	-	-	○
	add.a	%sp, %rs	sp(23:0)←sp(23:0)+rs(23:0)	-	-	-	-	-	-	○
		%rd, imm7	rd(23:0)←rd(23:0)+imm7(with zero extension)	-	-	-	-	-	-	○
	sadd.a	%sp, imm7	sp(23:0)←sp(23:0)+imm7(with zero extension)	-	-	-	-	-	-	○
		%rd, imm20	rd(23:0)←rd(23:0)+imm20(with zero extension)	-	-	-	-	-	-	-
	xadd.a	%sp, imm20	sp(23:0)←sp(23:0)+imm20(with zero extension)	-	-	-	-	-	-	-
		%rd, imm24	rd(23:0)←rd(23:0)+imm24	-	-	-	-	-	-	-
		%sp, imm24	sp(23:0)←sp(23:0)+imm24	-	-	-	-	-	-	-
		adc	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0)+C, rd(23:16)←0	-	-	↔	↔	↔	↔
	adc/c	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0)+C, rd(23:16)←0 if C = 1 (nop if C = 0)	-	-	↔	↔	↔	↔	○
	adc/nc	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0)+C, rd(23:16)←0 if C = 0 (nop if C = 1)	-	-	↔	↔	↔	↔	○
	adc	%rd, imm7	rd(15:0)←rd(15:0)+imm7(with zero extension)+C, rd(23:16)←0	-	-	↔	↔	↔	↔	○
	sadc	%rd, imm16	rd(15:0)←rd(15:0)+imm16+C, rd(23:16)←0	-	-	↔	↔	↔	↔	-
	xadc	%rd, imm16	rd(15:0)←rd(15:0)+imm16+C, rd(23:16)←0	-	-	↔	↔	↔	↔	-
	sub	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0), rd(23:16)←0	-	-	↔	↔	↔	↔	○
	sub/c	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	-	-	↔	↔	↔	↔	○
	sub/nc	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	-	-	↔	↔	↔	↔	○
	sub	%rd, imm7	rd(15:0)←rd(15:0)-imm7(with zero extension), rd(23:16)←0	-	-	↔	↔	↔	↔	○
	ssub	%rd, imm16	rd(15:0)←rd(15:0)-imm16, rd(23:16)←0	-	-	↔	↔	↔	↔	-
	xsub	%rd, imm16	rd(15:0)←rd(15:0)-imm16, rd(23:16)←0	-	-	↔	↔	↔	↔	-
	sub.a	%rd, %rs	rd(23:0)←rd(23:0)-rs(23:0)	-	-	-	-	-	-	○
	sub.a/c	%rd, %rs	rd(23:0)←rd(23:0)-rs(23:0) if C = 1 (nop if C = 0)	-	-	-	-	-	-	○
sub.a/nc	%rd, %rs	rd(23:0)←rd(23:0)-rs(23:0) if C = 0 (nop if C = 1)	-	-	-	-	-	-	○	
sub.a	%sp, %rs	sp(23:0)←sp(23:0)-rs(23:0)	-	-	-	-	-	-	○	
	%rd, imm7	rd(23:0)←rd(23:0)-imm7(with zero extension)	-	-	-	-	-	-	○	
	%sp, imm7	sp(23:0)←sp(23:0)-imm7(with zero extension)	-	-	-	-	-	-	○	
Remarks										



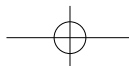
**Instruction List (6)****Assembly Programming**

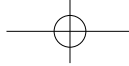
Classification	Mnemonic		Function	Flags					D	
	Opcode	Operand		IL	IE	C	V	Z		N
Arithmetic operation	ssub.a	%rd, imm20	rd(23:0)←rd(23:0)-imm20(with zero extension)	-	-	-	-	-	-	-
		%sp, imm20	sp(23:0)←sp(23:0)-imm20(with zero extension)	-	-	-	-	-	-	-
	xsub.a	%rd, imm24	rd(23:0)←rd(23:0)-imm24	-	-	-	-	-	-	-
		%sp, imm24	sp(23:0)←sp(23:0)-imm24	-	-	-	-	-	-	-
	sbc	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0)-C, rd(23:16)←0	-	-	↔	↔	↔	↔	○
	sbc/c	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0)-C, rd(23:16)←0 if C = 1 (nop if C = 0)	-	-	-	↔	↔	↔	○
	sbc/nc	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0)-C, rd(23:16)←0 if C = 0 (nop if C = 1)	-	-	-	↔	↔	↔	○
	sbc	%rd, imm7	rd(15:0)←rd(15:0)-imm7(with zero extension)-C, rd(23:16)←0	-	-	↔	↔	↔	↔	○
	ssbc	%rd, imm16	rd(15:0)←rd(15:0)-imm16-C, rd(23:16)←0	-	-	↔	↔	↔	↔	-
	xsbcc	%rd, imm16	rd(15:0)←rd(15:0)-imm16-C, rd(23:16)←0	-	-	↔	↔	↔	↔	-
	cmp	%rd, %rs	rd(15:0)-rs(15:0)	-	-	↔	↔	↔	↔	○
	cmp/c	%rd, %rs	rd(15:0)-rs(15:0) if C = 1 (nop if C = 0)	-	-	-	↔	↔	↔	○
	cmp/nc	%rd, %rs	rd(15:0)-rs(15:0) if C = 0 (nop if C = 1)	-	-	-	↔	↔	↔	○
	cmp	%rd, sign7	rd(15:0)-sign7(with sign extension)	-	-	↔	↔	↔	↔	○
	scmp	%rd, imm16	rd(15:0)-imm16	-	-	↔	↔	↔	↔	-
	xcmp	%rd, imm16	rd(15:0)-imm16	-	-	↔	↔	↔	↔	-
	cmp.a	%rd, %rs	d(23:0)-rs(23:0)	-	-	↔	↔	↔	↔	-
	cmp.a/c	%rd, %rs	rd(23:0)-rs(23:0) if C = 1 (nop if C = 0)	-	-	-	↔	↔	↔	○
	cmp.a/nc	%rd, %rs	rd(23:0)-rs(23:0) if C = 0 (nop if C = 1)	-	-	-	↔	↔	↔	○
	cmp.a	%rd, imm7	rd(23:0)-imm7(with zero extension)	-	-	↔	↔	↔	↔	○
	scmp.a	%rd, imm20	rd(23:0)-imm20(with zero extension)	-	-	↔	↔	↔	↔	-
	xcmp.a	%rd, imm24	rd(23:0)-imm24	-	-	↔	↔	↔	↔	-
	cmc	%rd, %rs	rd(15:0)-rs(15:0)-C	-	-	↔	↔	↔	↔	○
cmc/c	%rd, %rs	rd(15:0)-rs(15:0)-C if C = 1 (nop if C = 0)	-	-	-	↔	↔	↔	○	
cmc/nc	%rd, %rs	rd(15:0)-rs(15:0)-C if C = 0 (nop if C = 1)	-	-	-	↔	↔	↔	○	
cmc	%rd, sign7	rd(15:0)-sign7(with sign extension)-C	-	-	↔	↔	↔	↔	○	
scmc	%rd, imm16	rd(15:0)-imm16-C	-	-	↔	↔	↔	↔	-	
xcmc	%rd, imm16	rd(15:0)-imm16-C	-	-	↔	↔	↔	↔	-	
Logic operation	and	%rd, %rs	rd(15:0)←rd(15:0)&rs(15:0), rd(23:16)←0	-	-	-	0	↔	↔	○
	and/c	%rd, %rs	rd(15:0)←rd(15:0)&rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	-	-	-	0	↔	↔	○
	and/nc	%rd, %rs	rd(15:0)←rd(15:0)&rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	-	-	-	0	↔	↔	○
	and	%rd, sign7	rd(15:0)←rd(15:0)&sign7(with sign extension), rd(23:16)←0	-	-	-	0	↔	↔	○
	sand	%rd, imm16	rd(15:0)←rd(15:0)&imm16, rd(23:16)←0	-	-	-	0	↔	↔	-
	xand	%rd, imm16	rd(15:0)←rd(15:0)&imm16, rd(23:16)←0	-	-	-	0	↔	↔	-
Remarks										



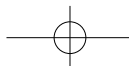
**Instruction List (7)****Assembly Programming**

Classification	Mnemonic		Function	Flags					D
	Opcode	Operand		IL	IE	C	V	Z	
Logic operation	or	%rd, %rs	$d(15:0) \leftarrow rd(15:0) \mid rs(15:0), rd(23:16) \leftarrow 0$	-	-	0	↔	↔	○
	or/c	%rd, %rs	$rd(15:0) \leftarrow rd(15:0) \mid rs(15:0), rd(23:16) \leftarrow 0$ if C = 1 (nop if C = 0)	-	-	0	↔	↔	○
	or/nc	%rd, %rs	$rd(15:0) \leftarrow rd(15:0) \mid rs(15:0), rd(23:16) \leftarrow 0$ if C = 0 (nop if C = 1)	-	-	0	↔	↔	○
	or	%rd, sign7	$rd(15:0) \leftarrow rd(15:0) \mid sign7(\text{with sign extension}), rd(23:16) \leftarrow 0$	-	-	0	↔	↔	○
	soor	%rd, imm16	$rd(15:0) \leftarrow rd(15:0) \mid imm16, rd(23:16) \leftarrow 0$	-	-	0	↔	↔	-
	xoor	%rd, imm16	$rd(15:0) \leftarrow rd(15:0) \oplus imm16, rd(23:16) \leftarrow 0$	-	-	0	↔	↔	-
	xor	%rd, %rs	$rd(15:0) \leftarrow rd(15:0) \wedge rs(15:0), rd(23:16) \leftarrow 0$	-	-	0	↔	↔	○
	xor/c	%rd, %rs	$rd(15:0) \leftarrow rd(15:0) \wedge rs(15:0), rd(23:16) \leftarrow 0$ if C = 1 (nop if C = 0)	-	-	0	↔	↔	○
	xor/nc	%rd, %rs	$rd(15:0) \leftarrow rd(15:0) \wedge rs(15:0), rd(23:16) \leftarrow 0$ if C = 0 (nop if C = 1)	-	-	0	↔	↔	○
	xor	%rd, sign7	$rd(15:0) \leftarrow rd(15:0) \wedge sign7(\text{with sign extension}), rd(23:16) \leftarrow 0$	-	-	0	↔	↔	○
	sxor	%rd, imm16	$rd(15:0) \leftarrow rd(15:0) \wedge imm16, rd(23:16) \leftarrow 0$	-	-	0	↔	↔	-
	xxor	%rd, imm16	$rd(15:0) \leftarrow rd(15:0) \wedge imm16, rd(23:16) \leftarrow 0$	-	-	0	↔	↔	-
	not	%rd, %rs	$rd(15:0) \leftarrow !rs(15:0), rd(23:16) \leftarrow 0$	-	-	0	↔	↔	○
	not/c	%rd, %rs	$rd(15:0) \leftarrow !rs(15:0), rd(23:16) \leftarrow 0$ if C = 1 (nop if C = 0)	-	-	0	↔	↔	○
	not/nc	%rd, %rs	$rd(15:0) \leftarrow !rs(15:0), rd(23:16) \leftarrow 0$ if C = 0 (nop if C = 1)	-	-	0	↔	↔	○
	not	%rd, sign7	$rd(15:0) \leftarrow !sign7(\text{with sign extension}), rd(23:16) \leftarrow 0$	-	-	0	↔	↔	○
snot	%rd, imm16	$rd(15:0) \leftarrow !imm16, rd(23:16) \leftarrow 0$	-	-	0	↔	↔	-	
xnot	%rd, imm16	$rd(15:0) \leftarrow !imm16, rd(23:16) \leftarrow 0$	-	-	0	↔	↔	-	
Branch	jpr / jpr.d	%rb sign10	$pc \leftarrow pc + 2 + rb$ $pc \leftarrow pc + 2 + sign11; sign11 = (sign10, 0)$	-	-	-	-	-	-
	sjpr / sjpr.d	label±imm20	$pc \leftarrow label \pm imm20$	-	-	-	-	-	-
		sign20	$pc \leftarrow pc + 2 + sign20$	-	-	-	-	-	-
	xjpr / xjpr.d	label±imm24	$pc \leftarrow label \pm imm24$	-	-	-	-	-	-
		sign24	$pc \leftarrow pc + 2 + sign24$	-	-	-	-	-	-
	jpa / jpa.d	%rb	$pc \leftarrow rb$	-	-	-	-	-	-
		imm7	$pc \leftarrow imm7$	-	-	-	-	-	-
	sjpa / sjpa.d	label±imm20	$pc \leftarrow label \pm imm20$	-	-	-	-	-	-
		imm20	$pc \leftarrow imm20$	-	-	-	-	-	-
	xjpa / xjpa.d	label±imm24	$pc \leftarrow label \pm imm24$	-	-	-	-	-	-
		imm24	$pc \leftarrow imm24$	-	-	-	-	-	-
	jrgt / jrgt.d	sign7	$pc \leftarrow pc + 2 + sign8$ if !Z&! (N^V) is true; sign8=(sign7,0)	-	-	-	-	-	-
	sjrgt / sjrgt.d	label±imm20	$pc \leftarrow label \pm imm20$ if !Z&! (N^V) is true	-	-	-	-	-	-
sign20		$pc \leftarrow pc + 2 + sign20$ if !Z&! (N^V) is true	-	-	-	-	-	-	
xjrgt / xjrgt.d	label±imm24	$pc \leftarrow label \pm imm24$ if !Z&! (N^V) is true	-	-	-	-	-	-	
	sign24	$pc \leftarrow pc + 2 + sign24$ if !Z&! (N^V) is true	-	-	-	-	-	-	
Remarks									

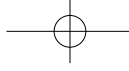


**Instruction List (8)****Assembly Programming**

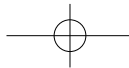
Classification	Mnemonic		Function	Flags						D	
	Opcode	Operand		IL	IE	C	V	Z	N		
Branch	<i>jrge / jrge.d</i>	<i>sign7</i> label±imm20	pc←pc+2+sign8 if !(N^V) is true; sign8={sign7,0}	-	-	-	-	-	-	-	
	<i>sjrge / sjrge.d</i>	sign20 label±imm20	pc←label±imm20 if !(N^V) is true pc←pc+2+sign20 if !(N^V) is true	-	-	-	-	-	-	-	
	<i>xjrge / xjrge.d</i>	<i>sign24</i> label±imm24	pc←label±imm24 if !(N^V) is true pc←pc+2+sign24 if !(N^V) is true	-	-	-	-	-	-	-	
	<i>jrlt / jrlt.d</i>	<i>sign7</i>	pc←pc+2+sign8 if N^V is true; sign8={sign7,0}	-	-	-	-	-	-	-	
	<i>sjrlt / sjrlt.d</i>	sign20 label±imm20	pc←label±imm20 if N^V is true pc←pc+2+sign20 if N^V is true	-	-	-	-	-	-	-	
	<i>xjrlt / xjrlt.d</i>	<i>sign24</i> label±imm24	pc←label±imm24 if N^V is true pc←pc+2+sign24 if N^V is true	-	-	-	-	-	-	-	
	<i>jrlz / jrlz.d</i>	<i>sign7</i>	pc←pc+2+sign8 if Z   (N^V) is true; sign8={sign7,0}	-	-	-	-	-	-	-	
	<i>sjrlz / sjrlz.d</i>	sign20 label±imm20	pc←label±imm20 if Z   (N^V) is true pc←pc+2+sign20 if Z   (N^V) is true	-	-	-	-	-	-	-	
	<i>xjrlz / xjrlz.d</i>	<i>sign24</i> label±imm24	pc←label±imm24 if Z   (N^V) is true pc←pc+2+sign24 if Z   (N^V) is true	-	-	-	-	-	-	-	
	<i>jrugt / jrugt.d</i>	<i>sign7</i>	pc←pc+2+sign8 if IZ&IC is true; sign8={sign7,0}	-	-	-	-	-	-	-	
	<i>sjrugt / sjrugt.d</i>	sign20 label±imm20	pc←label±imm20 if IZ&IC is true pc←pc+2+sign20 if IZ&IC is true	-	-	-	-	-	-	-	
	<i>xjrugt / xjrugt.d</i>	<i>sign24</i> label±imm24	pc←label±imm24 if IZ&IC is true pc←pc+2+sign24 if IZ&IC is true	-	-	-	-	-	-	-	
	<i>jruge / jruge.d</i>	<i>sign7</i>	pc←pc+2+sign8 if IC is true; sign8={sign7,0}	-	-	-	-	-	-	-	
	<i>sjruge / sjruge.d</i>	sign20 label±imm20	pc←label±imm20 if IC is true pc←pc+2+sign20 if IC is true	-	-	-	-	-	-	-	
	<i>xjruge / xjruge.d</i>	<i>sign24</i> label±imm24	pc←label±imm24 if IC is true pc←pc+2+sign24 if IC is true	-	-	-	-	-	-	-	
	<i>jrult / jrult.d</i>	<i>sign7</i>	pc←pc+2+sign8 if C is true; sign8={sign7,0}	-	-	-	-	-	-	-	
	<i>sjrult / sjrult.d</i>	sign20 label±imm20	pc←label±imm20 if C is true pc←pc+2+sign20 if C is true	-	-	-	-	-	-	-	
	<i>xjrult / xjrult.d</i>	<i>sign24</i> label±imm24	pc←label±imm24 if C is true pc←pc+2+sign24 if C is true	-	-	-	-	-	-	-	
	<i>jrulz / jrulz.d</i>	<i>sign7</i>	pc←pc+2+sign8 if Z   C is true; sign8={sign7,0}	-	-	-	-	-	-	-	
	<i>sjrulz / sjrulz.d</i>	sign20 label±imm20	pc←label±imm20 if Z   C is true pc←pc+2+sign20 if Z   C is true	-	-	-	-	-	-	-	
	<i>xjrulz / xjrulz.d</i>	<i>sign24</i> label±imm24	pc←label±imm24 if Z   C is true pc←pc+2+sign24 if Z   C is true	-	-	-	-	-	-	-	
	Remarks										

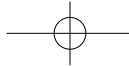




**Instruction List (9)****Assembly Programming**

Classification	Mnemonic		Function	Flags						D	
	Opcode	Operand		IL	IE	C	V	Z	N		
<b>Branch</b>	<i>jreq / jreq.d</i>	<i>sign7</i>	$pc \leftarrow pc + 2 + sign8$ if Z is true; $sign8 = (sign7, 0)$	-	-	-	-	-	-	-	
	<i>sjreq / sjreq.d</i>	$label \pm imm20$ <i>sign20</i>	$pc \leftarrow label \pm imm20$ if Z is true $pc \leftarrow pc + 2 + sign20$ if Z is false	-	-	-	-	-	-	-	
	<i>xjreq / xjreq.d</i>	$label \pm imm24$ <i>sign24</i>	$pc \leftarrow label \pm imm24$ if Z is true $pc \leftarrow pc + 2 + sign24$ if Z is false	-	-	-	-	-	-	-	
	<i>jrne / jrne.d</i>	<i>sign7</i>	$pc \leftarrow pc + 2 + sign8$ if IZ is true; $sign8 = (sign7, 0)$	-	-	-	-	-	-	-	
	<i>sjrne / sjrne.d</i>	$label \pm imm20$ <i>sign20</i>	$pc \leftarrow label \pm imm20$ if IZ is true $pc \leftarrow pc + 2 + sign20$ if IZ is false	-	-	-	-	-	-	-	
	<i>xjrne / xjrne.d</i>	$label \pm imm24$ <i>sign24</i>	$pc \leftarrow label \pm imm24$ if IZ is true $pc \leftarrow pc + 2 + sign24$ if IZ is false	-	-	-	-	-	-	-	
	<i>call / call.d</i>	<i>%rb</i> <i>sign10</i>	$sp \leftarrow sp - 4, A[sp] \leftarrow pc + 2(d=0)/4(d=1), pc \leftarrow pc + 2 + rb$ $sp \leftarrow sp - 4, A[sp] \leftarrow pc + 2(d=0)/4(d=1), pc \leftarrow pc + 2 + sign11; sign11 = (sign10, 0)$	-	-	-	-	-	-	-	
	<i>scall / scall.d</i>	$label \pm imm20$ <i>sign20</i>	$sp \leftarrow sp - 4, A[sp] \leftarrow pc + 2(d=0)/4(d=1), pc \leftarrow label \pm imm20$ $sp \leftarrow sp - 4, A[sp] \leftarrow pc + 2(d=0)/4(d=1), pc \leftarrow pc + 2 + sign20$	-	-	-	-	-	-	-	
	<i>xcall / xcall.d</i>	$label \pm imm24$ <i>sign24</i>	$sp \leftarrow sp - 4, A[sp] \leftarrow pc + 2(d=0)/4(d=1), pc \leftarrow label \pm imm24$ $sp \leftarrow sp - 4, A[sp] \leftarrow pc + 2(d=0)/4(d=1), pc \leftarrow pc + 2 + sign24$	-	-	-	-	-	-	-	
	<i>calla / calla.d</i>	<i>%rb</i> <i>imm7</i>	$sp \leftarrow sp - 4, A[sp] \leftarrow pc + 2(d=0)/4(d=1), pc \leftarrow rb$ $sp \leftarrow sp - 4, A[sp] \leftarrow pc + 2(d=0)/4(d=1), pc \leftarrow imm7$	-	-	-	-	-	-	-	
	<i>scalla / scalla.d</i>	$label \pm imm20$ <i>imm20</i>	$sp \leftarrow sp - 4, A[sp] \leftarrow pc + 2(d=0)/4(d=1), pc \leftarrow label \pm imm20$ $sp \leftarrow sp - 4, A[sp] \leftarrow pc + 2(d=0)/4(d=1), pc \leftarrow imm20$	-	-	-	-	-	-	-	
	<i>xcalla / xcalla.d</i>	$label \pm imm24$ <i>imm24</i>	$sp \leftarrow sp - 4, A[sp] \leftarrow pc + 2(d=0)/4(d=1), pc \leftarrow label \pm imm24$ $sp \leftarrow sp - 4, A[sp] \leftarrow pc + 2(d=0)/4(d=1), pc \leftarrow imm24$	-	-	-	-	-	-	-	
	<i>ret / ret.d</i>		$pc \leftarrow A[sp](23:0), sp \leftarrow sp + 4$	-	-	-	-	-	-	-	
	<i>int</i>	<i>imm5</i>	$sp \leftarrow sp - 4, A[sp] \leftarrow (psr, pc + 2), pc \leftarrow vector(TTBR + imm5 \times 4)$	-	0	-	-	-	-	-	
	<i>intl</i>	<i>imm5, imm3</i>	$sp \leftarrow sp - 4, A[sp] \leftarrow (psr, pc + 2), pc \leftarrow vector(TTBR + imm5 \times 4), psr(IL) \leftarrow imm3$	↔	0	-	-	-	-	-	
	<i>reti / reti.d</i>		$(psr, pc) \leftarrow A[sp], sp \leftarrow sp + 4$	↔	↔	↔	↔	↔	↔	-	
	<i>brk</i>		$A[DBRAM] \leftarrow (psr, pc + 2), A[DBRAM + 4] \leftarrow r0, pc \leftarrow 0xffffc00$	-	0	-	-	-	-	-	
	<i>retcd</i>		$r0 \leftarrow A[DBRAM + 4](23:0), (psr, pc) \leftarrow A[DBRAM]$	↔	↔	↔	↔	↔	↔	-	
	<b>Shift and swap</b>	<i>sr</i>	<i>%rd, %rs</i> <i>%rd, imm7</i>	Logical shift to right; $rd(15:0) \leftarrow rd(15:0) \gg rs(15:0), rd(23:16) \leftarrow 0$ , zero enters to MSB (*1) Logical shift to right; $rd(15:0) \leftarrow rd(15:0) \gg imm7, rd(23:16) \leftarrow 0$ , zero enters to MSB (*1)	-	-	↔	-	↔	↔	○
		<i>sa</i>	<i>%rd, %rs</i> <i>%rd, imm7</i>	Arithmetical shift to right; $rd(15:0) \leftarrow rd(15:0) \gg rs(15:0), rd(23:16) \leftarrow 0$ , sign copied to MSB (*1) Arithmetical shift to right; $rd(15:0) \leftarrow rd(15:0) \gg imm7, rd(23:16) \leftarrow 0$ , sign copied to MSB (*1)	-	-	↔	-	↔	↔	○
<i>sl</i>		<i>%rd, %rs</i> <i>%rd, imm7</i>	Logical shift to left; $rd(15:0) \leftarrow rd(15:0) \ll rs(15:0), rd(23:16) \leftarrow 0$ , zero enters to LSB (*1) Logical shift to left; $rd(15:0) \leftarrow rd(15:0) \ll imm7, rd(23:16) \leftarrow 0$ , zero enters to LSB (*1)	-	-	↔	-	↔	↔	○	
<i>swap</i>		<i>%rd, %rs</i>	$rd(15:8) \leftarrow rs(7:0), rd(7:0) \leftarrow rs(15:8), rd(23:16) \leftarrow 0$	-	-	-	-	-	-	○	

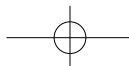
**Remarks**\*1) Number of bits to be shifted: Zero to three bits when  $rs/imm7 = 0-3$ , four bits when  $rs/imm7 = 4-7$ , eight bits when  $rs/imm7 \geq 8$ 

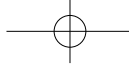


**Instruction List (10)** **Assembly Programming**

Classification	Mnemonic		Function	Flags					D	
	Opcode	Operand		IL	IE	C	V	Z		N
<b>Conversion</b>	cv.ab	%rd, %rs	rd(23:8)←rs(7), rd(7:0)←rs(7:0)	-	-	-	-	-	-	○
	cv.as	%rd, %rs	rd(23:16)←rs(15), rd(15:0)←rs(15:0)	-	-	-	-	-	-	○
	cv.al	%rd, %rs	rd(23:16)←rs(7:0), rd(15:0)←rd(15:0)	-	-	-	-	-	-	○
	cv.la	%rd, %rs	rd(23:8)←0, rd(7:0)←rs(23:16)	-	-	-	-	-	-	○
	cv.ls	%rd, %rs	rd(23:16)←0, rd(15:0)←rs(15)	-	-	-	-	-	-	○
<b>Imm extension</b>	ext	imm13	Extends the immediate or operand of the following instruction.	-	-	-	-	-	-	-
<b>System control</b>	nop		No operation	-	-	-	-	-	-	○
	halt		HALT mode	-	-	-	-	-	-	-
	slp		SLEEP mode	-	-	-	-	-	-	-
	ei		psr(IE)←1	-	1	-	-	-	-	○
	di		psr(IE)←0	-	0	-	-	-	-	○
<b>Coprocessor</b>	ld.cw	%rd, %rs	co_dout0←rd, co_dout1←rs	-	-	-	-	-	-	○
		%rd, imm7	co_dout0←rd, co_dout1←imm7	-	-	-	-	-	-	○
	sld.cw	%rd, imm20	co_dout0←rd, co_dout1←imm20	-	-	-	-	-	-	-
		%rd, symbol±imm20	co_dout0←rd, co_dout1←symbol±imm20	-	-	-	-	-	-	-
	xld.cw	%rd, imm24	co_dout0←rd, co_dout1←imm24	-	-	-	-	-	-	-
		%rd, symbol±imm24	co_dout0←rd, co_dout1←symbol±imm24	-	-	-	-	-	-	-
	ld.ca	%rd, %rs	co_dout0←rd, co_dout1←rs, rd←co_din, psr(C, V, Z, N)←co_cvzn	-	-	↔	↔	↔	↔	○
		%rd, imm7	co_dout0←rd, co_dout1←imm7, rd←co_din, psr(C, V, Z, N)←co_cvzn	-	-	↔	↔	↔	↔	○
	sld.ca	%rd, imm20	co_dout0←rd, co_dout1←imm20, rd←co_din, psr(C, V, Z, N)←co_cvzn	-	-	↔	↔	↔	↔	-
		%rd, symbol±imm20	co_dout0←rd, co_dout1←symbol±imm20, rd←co_din, psr(C, V, Z, N)←co_cvzn	-	-	↔	↔	↔	↔	-
	xld.ca	%rd, imm24	co_dout0←rd, co_dout1←imm24, rd←co_din, psr(C, V, Z, N)←co_cvzn	-	-	↔	↔	↔	↔	-
		%rd, symbol±imm24	co_dout0←rd, co_dout1←symbol±imm24, rd←co_din, psr(C, V, Z, N)←co_cvzn	-	-	↔	↔	↔	↔	-
	ld.cf	%rd, %rs	co_dout0←rd, co_dout1←rs, psr(C, V, Z, N)←co_cvzn	-	-	↔	↔	↔	↔	○
		%rd, imm7	co_dout0←rd, co_dout1←imm7, psr(C, V, Z, N)←co_cvzn	-	-	↔	↔	↔	↔	○
	sld.cf	%rd, imm20	co_dout0←rd, co_dout1←imm20, psr(C, V, Z, N)←co_cvzn	-	-	↔	↔	↔	↔	-
%rd, symbol±imm20		co_dout0←rd, co_dout1←symbol±imm20, psr(C, V, Z, N)←co_cvzn	-	-	↔	↔	↔	↔	-	
xld.cf	%rd, imm24	co_dout0←rd, co_dout1←imm24, psr(C, V, Z, N)←co_cvzn	-	-	↔	↔	↔	↔	-	
	%rd, symbol±imm24	co_dout0←rd, co_dout1←symbol±imm24, psr(C, V, Z, N)←co_cvzn	-	-	↔	↔	↔	↔	-	

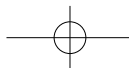
**Remarks**





**Expansion Format of Extended Instructions (1) Assembly Programming**

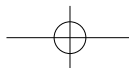
Extended instruction		Expansion format		
Opcode	Operand	Condition 1	Condition 2	Condition 3
sld.b sld.ub sld sld.a	%rd, [%sp+imm20]	imm20≤0x7f	0x7f<imm20	–
	Example) sld.b %rd, [%sp+imm20]	ld.b %rd, [%sp+imm20(6:0)]	ext imm20(19:7)	
			ld.b %rd, [%sp+imm20(6:0)]	
	%rd, [imm20]	imm20≤0x7f	0x7f<imm20	–
Example) sld %rd, [imm20]		ld %rd, [imm20(6:0)]	ext imm20(19:7)	
			ld %rd, [imm20(6:0)]	
sld.b sld sld.a	[%sp+imm20], %rs	imm20≤0x7f	0x7f<imm20	–
	Example) sld.b [%sp+imm20], %rs	ld.b [%sp+imm20(6:0)], %rs	ext imm20(19:7)	
			ld.b [%sp+imm20(6:0)], %rs	
	[imm20], %rs	imm20≤0x7f	0x7f<imm20	–
Example) sld [imm20], %rs		ld [imm20(6:0)], %rs	ext imm20(19:7)	
			ld [imm20(6:0)], %rs	
sld	%rd, imm16	imm16≤0x7f	0x7f<imm16	–
	Example) sld %rd, imm16	ld %rd, imm16(6:0)	ext imm16(15:7)	
			ld %rd, imm16(6:0)	
	%rd, symbol±imm16	Unconditional	–	–
Example) sld %rd, symbol+imm16	ext (symbol+imm16)(15:7)			
	ld %rd, (symbol+imm16)(6:0)			
sld.a	%rd, imm20	imm20≤0x7f	0x7f<imm20	–
	Example) sld.a %rd, imm20	ld.a %rd, imm20(6:0)	ext imm20(19:7)	
			ld.a %rd, imm20(6:0)	
	%sp, imm20	imm20≤0x7f	0x7f<imm20	–
Example) sld.a %sp, imm20	ld.a %sp, imm20(6:0)	ext imm20(19:7)		
			ld.a %sp, imm20(6:0)	
<b>Remarks</b>				





**Expansion Format of Extended Instructions (2) Assembly Programming**

Extended instruction		Expansion format		
Opcode	Operand	Condition 1	Condition 2	Condition 3
sld.a	%rd, symbol±imm20  Example) sld.a %rd, symbol+imm20	Unconditional	–	–
		ext (symbol+imm20)(19:7) ld.a %rd, (symbol+imm20)(6:0)		
	%sp, symbol±imm20  Example) sld.a %sp, symbol-imm20	Unconditional	–	–
		ext (symbol-imm20)(19:7) ld.a %sp, (symbol-imm20)(6:0)		
xld.b xld.ub xld xld.a	%rd, [%sp+imm24]  Example) xld.b %rd, [%sp+imm24]	imm24≤0x7f	0x7f<imm24≤0xffff	0xffff<imm24
		ld.b %rd, [%sp+imm24(6:0)]	ext imm24(19:7) ld.b %rd, [%sp+imm24(6:0)]	ext imm24(23:20) ext imm24(19:7) ld.b %rd, [%sp+imm24(6:0)]
	%rd, [imm24]  Example) xld %rd, [imm24]	imm24≤0x7f	0x7f<imm24≤0xffff	0xffff<imm24
		ld %rd, [imm24(6:0)]	ext imm24(19:7) ld %rd, [imm24(6:0)]	ext imm24(23:20) ext imm24(19:7) ld %rd, [imm24(6:0)]
xld.b xld xld.a	[%sp+imm24], %rs  Example) xld.b [%sp+imm24], %rs	imm24≤0x7f	0x7f<imm24≤0xffff	0xffff<imm24
		ld.b [%sp+imm24(6:0)], %rs	ext imm24(19:7) ld.b [%sp+imm24(6:0)], %rs	ext imm24(23:20) ext imm24(19:7) ld.b [%sp+imm24(6:0)], %rs
	[imm24], %rs  Example) xld [imm24], %rs	imm24≤0x7f	0x7f<imm24≤0xffff	0xffff<imm24
		ld [imm24(6:0)], %rs	ext imm24(19:7) ld [imm24(6:0)], %rs	ext imm24(23:20) ext imm24(19:7) ld [imm24(6:0)], %rs
xld	%rd, imm16  Example) xld %rd, imm16	imm16≤0x7f	0x7f<imm16	–
		ld %rd, imm16(6:0)	ext imm16(15:7) ld %rd, imm16(6:0)	
	%rd, symbol±imm16  Example) xld %rd, symbol+imm16	Unconditional	–	–
		ext (symbol+imm16)(15:7) ld %rd, (symbol+imm16)(6:0)		
<b>Remarks</b>				



**Expansion Format of Extended Instructions (3)**

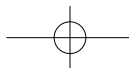
**Assembly Programming**

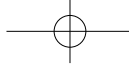
Extended instruction		Expansion format			
Opcode	Operand	Condition 1	Condition 2	Condition 3	
<b>xld.a</b>	<b>%rd, imm24</b>  Example) xld.a %rd, imm24	imm24≤0x7f	0x7f<imm24≤0xffff	0xffff<imm24	
		ld.a %rd, imm24(6:0)	ext imm24(19:7) ld.a %rd, imm24(6:0)	ext imm24(23:20) ext imm24(19:7) ld.a %rd, imm24(6:0)	
	<b>%sp, imm24</b>  Example) xld.a %sp, imm24	imm24≤0x7f	0x7f<imm24≤0xffff	0xffff<imm24	
		ld.a %sp, imm24(6:0)	ext imm24(19:7) ld.a %sp, imm24(6:0)	ext imm24(23:20) ext imm24(19:7) ld.a %sp, imm24(6:0)	
	<b>%rd, symbol±imm24</b>  Example) xld.a %rd, symbol+imm24	Unconditional	–	–	
		ext (symbol+imm24)(23:20) ext (symbol+imm24)(19:7) ld.a %rd, (symbol+imm24)(6:0)			
		<b>%sp, symbol±imm24</b>  Example) xld.a %sp, symbol-imm24	Unconditional	–	–
	ext (symbol-imm24)(23:20) ext (symbol-imm24)(19:7) ld.a %sp, (symbol-imm24)(6:0)				
	<b>sadd</b> <b>sadc</b> <b>ssub</b> <b>ssbc</b>  Example) sadd %rd, imm16		imm16≤0x7f	0x7f<imm16	–
		add %rd, imm16(6:0)	ext imm16(15:7) add %rd, imm16(6:0)		
	<b>sadd.a</b> <b>ssub.a</b>	<b>%rd, imm20</b>  Example) ssub.a %rd, imm20	imm20≤0x7f	0x7f<imm20	–
			sub.a %rd, imm20(6:0)	ext imm20(19:7) sub.a %rd, imm20(6:0)	
<b>%sp, imm20</b>  Example) sadd.a %sp, imm20		imm20≤0x7f	0x7f<imm20	–	
		add.a %sp, imm20(6:0)	ext imm20(19:7) add.a %sp, imm20(6:0)		
<b>xadd</b> <b>xadc</b> <b>xsub</b> <b>xsbc</b>  Example) xadc %rd, imm16	imm16≤0x7f	0x7f<imm16	–		
	adc %rd, imm16(6:0)	ext imm16(15:7) adc %rd, imm16(6:0)			
	<b>Remarks</b>				

**Expansion Format of Extended Instructions (4)****Assembly Programming**

Extended instruction		Expansion format		
Opcode	Operand	Condition 1	Condition 2	Condition 3
<b>xadd.a</b> <b>xsub.a</b>	%rd, imm24  Example) xsub.a %rd, imm24	imm24≤0x7f	0x7f<imm24≤0xffff	0xffff<imm24
		sub.a %rd, imm24(6:0)	ext imm24(19:7) sub.a %rd, imm24(6:0)	ext imm24(23:20) ext imm24(19:7) sub.a %rd, imm24(6:0)
	%sp, imm24  Example) xadd.a %sp, imm24	imm24≤0x7f	0x7f<imm24≤0xffff	0xffff<imm24
		add.a %sp, imm24(6:0)	ext imm24(19:7) add.a %sp, imm24(6:0)	ext imm24(23:20) ext imm24(19:7) add.a %sp, imm24(6:0)
<b>scmp</b> <b>scmc</b>	%rd, imm16  Example) scmp %rd, imm16	imm16≤0x7f	0x7f<imm16	–
		cmp %rd, imm16(6:0)	ext imm16(15:7) cmp %rd, imm16(6:0)	
<b>scmp.a</b>	%rd, imm20  Example) scmp.a %rd, imm20	imm20≤0x7f	0x7f<imm20	–
		cmp.a %rd, imm20(6:0)	ext imm20(19:7) cmp.a %rd, imm20(6:0)	
<b>xcmp</b> <b>xcmc</b>	%rd, imm16  Example) xcmc %rd, imm16	imm16≤0x7f	0x7f<imm16	–
		cmc %rd, imm16(6:0)	ext imm16(15:7) cmc %rd, imm16(6:0)	
<b>xcmp.a</b>	%rd, imm24  Example) xcmp.a %rd, imm24	imm24≤0x7f	0x7f<imm24≤0xffff	0xffff<imm24
		cmp.a %rd, imm24(6:0)	ext imm24(19:7) cmp.a %rd, imm24(6:0)	ext imm24(23:20) ext imm24(19:7) cmp.a %rd, imm24(6:0)
<b>sand</b> <b>soor</b> <b>sxor</b> <b>snot</b>	%rd, imm16  Example) sand %rd, imm16	imm16≤0x7f	0x7f<imm16	–
		and %rd, imm16(6:0)	ext imm16(15:7) and %rd, imm16(6:0)	
<b>xand</b> <b>xoor</b> <b>xxor</b> <b>xnot</b>	%rd, imm16  Example) xoor %rd, imm16	imm16≤0x7f	0x7f<imm16	–
		or %rd, imm16(6:0)	ext imm16(15:7) or %rd, imm16(6:0)	

Remarks



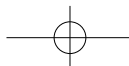


**Expansion Format of Extended Instructions (5)** **Assembly Programming**

Extended instruction		Expansion format		
Opcode	Operand	Condition 1	Condition 2	Condition 3
<b>scall</b> <b>scall.d</b> <b>sjpr</b> <b>sjpr.d</b>	<b>label±imm20</b>  Example) scall label+imm20	Unconditional	–	–
		ext (label+imm20)(19:12) call (label+imm20)(11:1)		
	<b>sign20</b>  Example) sjpr sign20	$-1024 \leq \text{sign20} \leq 1023$	$\text{sign20} < -1024$ or $1023 < \text{sign20}$	–
		jpr sign20(11:1)	ext sign20(19:12) jpr sign20(11:1)	
<b>sjr*1</b> <b>sjr*1.d</b>	<b>label±imm20</b>  Example) sjreq label+imm20	Unconditional	–	–
		ext (label+imm20)(19:8) jreq (label+imm20)(7:1)		
	<b>sign20</b>  Example) sjrne sign20	$-128 \leq \text{sign20} \leq 127$	$\text{sign20} < -128$ or $127 < \text{sign20}$	–
		jrne sign20(7:1)	ext sign20(19:8) jrne sign20(7:1)	
<b>scalla</b> <b>scalla.d</b> <b>sja</b> <b>sja.d</b>	<b>label±imm20</b>  Example) scalla label+imm20	Unconditional	–	–
		ext (label+imm20)(19:7) calla (label+imm20)(6:0)		
	<b>imm20</b>  Example) sja imm20	$\text{imm20} \leq 0x7f$	$0x7f < \text{imm20}$	–
		jpa imm20(6:0)	ext imm20(19:7) jpa imm20(6:0)	
<b>xcall</b> <b>xcall.d</b> <b>xjpr</b> <b>xjpr.d</b>	<b>label±imm24</b>  Example) xcall label+imm24	Unconditional	–	–
		ext (label+imm24)(23:12) call (label+imm24)(11:1)		
	<b>sign24</b>  Example) xjpr sign24	$-1024 \leq \text{sign24} \leq 1023$	$\text{sign24} < -1024$ or $1023 < \text{sign24}$	–
		jpr sign24(11:1)	ext sign24(23:12) jpr sign24(11:1)	

**Remarks**

\*1) sjreq, sjreq.d, sjrne, sjrne.d, sjrgt, sjrgt.d, sjrge, sjrge.d, sjrft, sjrft.d, sjrle, sjrle.d, sjrugt, sjrugt.d, sjruge, sjruge.d, sjrult, sjrult.d, sjrule, sjrule.d



**Expansion Format of Extended Instructions (6)**

**Assembly Programming**

Extended instruction		Expansion format		
Opcode	Operand	Condition 1	Condition 2	Condition 3
xjr*1 xjr*1.d	label±imm24  Example) xjreq label+imm24	Unconditional	–	–
		ext (label+imm24)(23:21)		
		ext (label+imm24)(20:8)		
	sign24	-128≤sign24≤127	-1048576≤sign24<-128 or 127<sign24≤1048575	sign24<-1048576 or 1048575<sign24
	jrne sign24(7:1)  Example) xjrne sign24	ext sign24(20:8) jrne sign24(7:1)	ext sign24(23:21) ext sign24(20:8) jrne sign24(7:1)	
xcalla xcalla.d xjpa xjpa.d	label±imm24  Example) xcalla label+imm24	Unconditional	–	–
		ext (label+imm24)(23:20)		
		ext (label+imm24)(19:7)		
	imm24	imm24≤0x7f	0x7f<imm24≤0xffff	0xffff<imm24
	jpa imm24(6:0)  Example) xjpa imm24	ext imm24(19:7) jpa imm24(6:0)	ext imm24(23:20) ext imm24(19:7) jpa imm24(6:0)	
sld.cw sld.ca sld.cf	%rd, imm20  Example) sld.cw %rd, imm20	imm20≤0x7f	0x7f<imm20	–
		ld.cw %rd, imm20(6:0)	ext imm20(19:7) ld.cw %rd, imm20(6:0)	
		Unconditional	–	–
	%rd, symbol±imm20  Example) sld.ca %rd, symbol+imm20	ext (symbol+imm20)(19:7) ld.ca %rd, (symbol+imm20)(6:0)		
xld.cw xld.ca xld.cf	%rd, imm24  Example) xld.cw %rd, imm24	imm24≤0x7f	0x7f<imm24≤0xffff	0xffff<imm24
		ld.cw %rd, imm24(6:0)	ext imm24(19:7) ld.cw %rd, imm24(6:0)	ext imm24(23:20) ext imm24(19:7) ld.cw %rd, imm24(6:0)
		Unconditional	–	–
	%rd, symbol±imm24  Example) xld.ca %rd, symbol+imm24	ext (symbol+imm24)(23:20) ext (symbol+imm24)(19:7) ld.ca %rd, (symbol+imm24)(6:0)		
<b>Remarks</b>				
*1) xjreq, xjreq.d, xjrne, xjrne.d, xjrgt, xjrgt.d, xjrge, xjrge.d, xjrft, xjrft.d, xjrle, xjrle.d, xjrugt, xjrugt.d, xjruge, xjruge.d, xjrult, xjrult.d, xjrule, xjrule.d				



## AMERICA

---

### EPSON ELECTRONICS AMERICA, INC.

#### HEADQUARTERS

2580 Orchard Parkway  
San Jose, CA 95131, U.S.A.  
Phone: +1-800-228-3964 Fax: +1-408-922-0238

#### SALES OFFICE

##### Northeast

301 Edgewater Place, Suite 210  
Wakefield, MA 01880, U.S.A.  
Phone: +1-800-922-7667 Fax: +1-781-246-5443

## EUROPE

---

### EPSON EUROPE ELECTRONICS GmbH

#### HEADQUARTERS

Riesstrasse 15  
80992 Munich, GERMANY  
Phone: +49-89-14005-0 Fax: +49-89-14005-110

## ASIA

---

### EPSON (CHINA) CO., LTD.

23F, Beijing Silver Tower 2# North RD DongSanHuan  
ChaoYang District, Beijing, CHINA  
Phone: +86-10-6410-6655 Fax: +86-10-6410-7320

#### SHANGHAI BRANCH

7F, High-Tech Bldg., 900, Yishan Road  
Shanghai 200233, CHINA  
Phone: +86-21-5423-5522 Fax: +86-21-5423-5512

### EPSON HONG KONG LTD.

20/F, Harbour Centre, 25 Harbour Road  
Wanchai, Hong Kong  
Phone: +852-2585-4600 Fax: +852-2827-4346  
Telex: 65542 EPSCO HX

### EPSON Electronic Technology Development (Shenzhen) LTD.

12/F, Dawning Mansion, Keji South 12th Road  
Hi-Tech Park, Shenzhen  
Phone: +86-755-2699-3828 Fax: +86-755-2699-3838

### EPSON TAIWAN TECHNOLOGY & TRADING LTD.

14F, No. 7, Song Ren Road  
Taipei 110  
Phone: +886-2-8786-6688 Fax: +886-2-8786-6660

### EPSON SINGAPORE PTE., LTD.

1 HarbourFront Place  
#03-02 HarbourFront Tower One, Singapore 098633  
Phone: +65-6586-5500 Fax: +65-6271-3182

### SEIKO EPSON CORPORATION

#### KOREA OFFICE

50F, KLI 63 Bldg., 60 Yoido-dong  
Youngdeungpo-Ku, Seoul, 150-763, KOREA  
Phone: +82-2-784-6027 Fax: +82-2-767-3677

#### GUMI OFFICE

2F, Grand B/D, 457-4 Songjeong-dong  
Gumi-City, KOREA  
Phone: +82-54-454-6027 Fax: +82-54-454-6093

### SEIKO EPSON CORPORATION SEMICONDUCTOR OPERATIONS DIVISION

#### IC Sales Dept.

#### IC International Sales Group

421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN  
Phone: +81-42-587-5814 Fax: +81-42-587-5117

**S5U1C17001C Manual**  
(C Compiler Package for S1C17 Family) (Ver. 1.1)

**SEIKO EPSON CORPORATION**  
**SEMICONDUCTOR OPERATIONS DIVISION**

■ EPSON Electronic Devices Website

[http://www.epson.jp/device/semicon\\_e](http://www.epson.jp/device/semicon_e)