EPSON
EXCEED YOUR VISION

CMOS 32-BIT SINGLE CHIP MICROCOMPUTER

S5U1C33001C Manual

(C/C++ Compiler Package for S1C33 Family)
(Ver. 3.3.0)

SEIKO EPSON CORPORATION

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, Windows XP, and Windows Vista 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** 2009, All rights reserved.

Configuration of product number

Devices

S1 C 33209 F 00E1 00

Packing specifications

[00 : Besides tape & reel
0A : TCP BL 2 directions
0B : Tape & reel BACK
0C : TCP BR 2 directions
0D : TCP BT 2 directions
0E : TCP BD 2 directions
0F : Tape & reel FRONT
0G : TCP BT 4 directions
0H : TCP BD 4 directions
0J : TCP SL 2 directions
0K : TCP SR 2 directions
0L : Tape & reel LEFT
0M : TCP ST 2 directions
0N : TCP SD 2 directions
0P : TCP ST 4 directions
0Q : TCP SD 4 directions
0R : Tape & reel RIGHT
99 : Specs not fixed]

Specification

Package
[D: die form; F: QFP, B: BGA]

Model number

Model name
[C: microcomputer, digital products]

Product classification
[S1: semiconductor]

Development tools

S5U1 C 33000 H2 1 00

Packing specifications
[00: standard packing]

Version
[1: Version 1]

Tool type

[Hx : ICE
Dx : Evaluation board
Ex : ROM emulation board
Mx : Emulation memory for external ROM
Tx : A socket for mounting
Cx : Compiler package
Sx : Middleware package]

Corresponding model number
[33L01: for S1C33L01]

Tool classification
[C: microcomputer use]

Product classification
[S5U1: development tool for semiconductor products]

Introduction

This document describes the development procedure from compiling C/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 S1C33 Family C/C++ Compiler Package common to all the models of the S1C33 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/C++ language (based on ANSI C) and C/C++ source creation methods
- Knowledge about the gnu C/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/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/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:

S1C33*xxx* Technical Manual
  Covers device specifications, and the operation and control method of the peripheral circuits.
S1C33000 Core CPU 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:

S1C33 Family In-Circuit Debugger Manual
  Explains the functions and handling methods of the S5U1C3300*x*H in-circuit debugger.
S1C33 Family Debug Monitor Operation Manual
  Explains the functions and implementation of the S5U1C330M2S debug monitor.

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 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.

– Contents –

1 General 1-1
1.1 Features 1-1
1.2 Tool Composition 1-2

2 Installation 2-1
2.1 Working Environment 2-1
2.2 Installation Method 2-2

3 Software Development Procedures 3-1
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 3-17
  3.3.5 Building a Program 3-25
  3.3.6 Debugging a Program 3-26
  3.3.7 Creating ROM Data 3-40
3.4 Tutorial 2 (Using the User Makefiles) 3-41
  3.4.1 Creating a Project 3-41
  3.4.2 Importing Source Files 3-43
  3.4.3 Disabling the GNU33 File Builder 3-45
  3.4.4 Setting and Correcting the Makefile 3-46
  3.4.5 Building a Project 3-48
  3.4.6 Importing Debugger Startup Files 3-49
  3.4.7 Correcting Debugger Startup Files 3-51
  3.4.8 Starting the Debugger 3-52
3.5 Tutorial 3 (Importing an IDE Project) 3-54
3.6 Debugging Environment 3-58
  3.6.1 Debug Monitor S5U1C330M2S 3-58
  3.6.2 In-Circuit Debugger S5U1C33000H 3-61
  3.6.3 In-Circuit Debugger S5U1C33001H 3-63
3.7 Data Area and Sections 3-65
  3.7.1 Data Area 3-65
  3.7.2 Sections 3-68

4 Source Files 4-1
4.1 File Format and File Name 4-1
4.2 Grammar of C/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
  4.2.5 `alloca()`/Variable Length Array 4-5
4.3 Grammar of Assembly Source 4-6
  4.3.1 Statements 4-6
  4.3.2 Notations of Operands 4-10
  4.3.3 Extended Instructions 4-14
  4.3.4 Preprocessor Directives 4-15
4.4 Precautions for Creation of Sources 4-16

5 GNU33 IDE....................5-1
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....................5-46
5.4.7 Changing the Project Name....................5-47
5.4.8 Resource Manipulation in a Project....................5-48
5.4.9 File Filter....................5-69
5.4.10 Working Set....................5-70
5.4.11 Project Properties....................5-74
5.5 The Editor and Editing Source Files....................5-76
5.5.1 Starting the Editor....................5-76
5.5.2 Basic Editing Facilities....................5-78
5.5.3 Editing Functions for C/C++ Source Files....................5-79
5.5.4 [Outline] View....................5-86
5.5.5 Navigation History....................5-87
5.5.6 Bookmarks....................5-88
5.5.7 Tasks....................5-93
5.5.8 Customizing the Editor....................5-99
5.5.9 Using an External Editor....................5-101
5.5.10 Launching External Editor by Specifying Line Number....................5-103
5.6 Search....................5-105
5.6.1 Text Search....................5-105
5.6.2 File Search....................5-105
5.6.3 C/C++ Search....................5-107
5.6.4 C/C++ Search from Context Menu....................5-108

5.6.5 Canceling a Search 5-108
5.6.6 Search Results 5-109
5.7 Building a Program..5-111
5.7.1 Setting the CPU Type..5-111
5.7.2 Selecting Two-Pass/One-Pass Make 5-113
5.7.3 Setting Compiler Options..5-114
5.7.4 Setting Assembler Options 5-121
5.7.5 Setting Linker Options..5-123
5.7.6 Generated Makefile..5-127
5.7.7 Editing a Linker Script..5-130
5.7.8 Executing a Build Process 5-158
5.7.9 Clean and Rebuild 5-160
5.7.10 Using an Original Makefile..5-162
5.8 Starting the Debugger..5-164
5.8.1 Generating a Parameter File..5-164
5.8.2 Setting the Debugger Startup Commands..5-169
5.8.3 Starting Up the Debugger 5-172
5.9 Customizing the IDE (Preferences)..5-175
5.10 Additional Description on Dialog Boxes 5-206
5.10.1 Properties for Project 5-206
5.10.2 Save Resources..5-237
5.10.3 New C++ Class..5-238
5.10.4 Choose Base Class 5-240
5.10.5 Import > File system 5-241
5.10.6 Export > File system 5-243
5.10.7 Filters 5-245
5.11 Files Generated in a Project by the IDE..5-247

6 C/C++ Compiler..6-1

6.1 Functions..6-1
6.2 Input/Output Files..6-1
6.2.1 Input Files 6-1
6.2.2 Output Files 6-2
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 6-9
6.4.1 Output Contents 6-9
6.4.2 Data Representation..6-10
6.4.3 Method of Using Registers 6-12
6.4.4 Function Call..6-14
6.4.5 Stack Frame..6-15
6.4.6 Grammar of C/C++ Source 6-15
6.5 Standard Header File and Standard Header..6-16
6.6 Points to Note when Using C++ 6-18
6.7 Upgrading the gcc Core and Added/Removed Features..6-20
6.8 Filter Function for Shift JIS Code 6-21
6.9 C and Assembler Mixed Programming..6-22
6.10 Functions of **xgcc** and Usage Precautions 6-27
6.11 Known issues 6-28

7 Library 7-1
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 Remainder Calculation Functions 7-6
7.3 `long long` Type Emulation Library..7-7
7.3.1 Overview 7-7
7.3.2 `long long` Type Calculation Functions..7-7
7.4 ANSI Library..7-8
7.4.1 Overview 7-8
7.4.2 ANSI Library Function List 7-8
7.4.3 Declaring and Initializing Global Variables 7-14
7.4.4 Lower-level Functions 7-15
7.5 C++ Library 7-17
7.5.1 Overview 7-17
7.5.2 List of STL Container Member Functions 7-19
8 Assembler 8-1
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 8-5
8.5 Assembler Directives..8-6
8.5.1 Text Section Defining Directive (`.text`) 8-6
8.5.2 Data Section Defining Directives (`.rodata`, `.data`) 8-7
8.5.3 Bss Section Defining Directive (`.bss`)..8-8
8.5.4 Comm Section Defining Directive (`.comm`) 8-9
8.5.5 Ctors Section Defining Directive (`.ctors`) 8-10
8.5.6 Dtors Section Defining Directive (`.dtors`) 8-11
8.5.7 Gcc_except_table Section Defining Directive (`.gcc_except_table`) 8-12
8.5.8 Data Defining Directives (`.long`, `.short`, `.byte`, `.ascii`, `.space`)..........8-13
8.5.9 Area Securing Directive (`.zero`) 8-14
8.5.10 Alignment Directive (`.align`)..8-15
8.5.11 Global Declaring Directive (`.global`)..8-16
8.5.12 Symbol Defining Directive (`.set`)..8-17
8.6 Pseudo-operands..8-18
8.7 Extended Instructions..8-19
8.7.1 Arithmetic Operation Instructions..8-19
8.7.2 Comparison Instructions 8-20
8.7.3 Logic Operation Instructions 8-21
8.7.4 Shift & Rotate Instructions 8-22
8.7.5 Data Transfer Instructions (between Stack and Register) 8-23
8.7.6 Data Transfer Instructions (between Memory and Register)..........................8-24
8.7.7 Immediate Data Load Instructions 8-26
8.7.8 Bit Operation Instructions 8-27
8.7.9 Branch Instructions 8-28

8.7.10 C33 ADV Core Data Transfer Instructions (between Memory and Register) ..8-31
8.7.11 C33 ADV Core Data Transfer Instructions (between `%dp` Area and Register) 8-32
8.8 Optimization of Extended Instructions....8-33
8.9 Error/Warning Messages 8-39
8.10 Precautions 8-40

9 Linker....9-1
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....9-3
9.3.1 Startup Format....9-3
9.3.2 Command-line Options 9-3
9.4 Linkage....9-4
9.4.1 Specifying Data Area Pointer....9-4
9.4.2 Default Linker Script....9-4
9.4.3 Examples of Linkage 9-7
9.5 Error/Warning Messages 9-11
9.6 Precautions 9-12

10 Debugger....10-1
10.1 Features 10-1
10.2 Input/Output Files....10-1
10.2.1 Input Files 10-2
10.2.2 Output Files 10-3
10.3 Starting the Debugger....10-4
10.3.1 Startup Format 10-4
10.3.2 Startup Options....10-4
10.3.3 Quitting the Debugger....10-5
10.4 Windows....10-6
10.4.1 Basic Window Configuration 10-6
10.4.2 [Source] Window....10-7
10.4.3 [Console] Window 10-20
10.4.4 [Registers] Window 10-21
10.4.5 [Memory] Window 10-24
10.4.6 [Breakpoints] Window 10-27
10.4.7 [Watch Expressions] Window....10-29
10.4.8 [Local Variables] Window 10-32
10.4.9 [Simulated I/O] Window....10-34
10.4.10 [Trace] Window....10-35
10.5 Method of Executing Commands 10-37
10.5.1 Entering Commands From the Keyboard....10-37
10.5.2 Parameter Input Format....10-38
10.5.3 Using Menus and Toolbar To Execute Commands 10-39
10.5.4 Using a Command File To Execute Commands 10-41
10.5.5 Log Files 10-42
10.6 Debugging Functions 10-43
10.6.1 Connect modes....10-43
10.6.2 Loading a File 10-47
10.6.3 Source-Level Debugging Function....10-48
10.6.4 Manipulating Memory, Variables, and Registers 10-49
10.6.5 Executing the Program 10-53

10.6.6 Break Functions 10-58
10.6.7 Trace Functions 10-65
10.6.8 Simulated I/O 10-73
10.6.9 Flash Memory Operation 10-75
10.6.10 Support for Big Endian 10-78
10.7 Command Reference 10-79
10.7.1 List of Commands 10-79
10.7.2 Detailed Description of Commands 10-80
10.7.3 Memory Manipulation Commands 10-81
c33 fb (fill area, in bytes) 10-81
c33 fh (fill area, in half words) 10-81
c33 fw (fill area, in words) 10-81
x (memory dump) 10-83
set { } (data input) 10-85
c33 mvb (copy area, in bytes) 10-86
c33 mvh (copy area, in half words) 10-86
c33 mvw (copy area, in words) 10-86
c33 df (save memory contents) 10-88
c33 rm (read target memory) 10-90
c33 readmd (memory read mode) 10-91
10.7.4 Register Manipulation Commands 10-92
info reg (display register) 10-92
set $ (modify register) 10-95
10.7.5 Program Execution Commands 10-96
continue (execute continuously) 10-96
until (execute continuously with temporary break) 10-98
step (single-step, every line) 10-100
stepi (single-step, every mnemonic) 10-100
next (single-step with skip, every line) 10-102
nexti (single-step with skip, every mnemonic) 10-102
finish (finish function) 10-104
c33 callmd (set user function call mode) 10-105
c33 call (call user function) 10-106
10.7.6 CPU Reset Commands 10-108
c33 rstc (cold reset) 10-108
c33 rsth (hot reset) 10-110
c33 rstt (reset target) 10-111
10.7.7 Interrupt Command 10-112
c33 int (interrupt) 10-112
10.7.8 Break Setup Commands 10-113
break (set software PC break) 10-113
tbreak (set temporary software PC break) 10-113
hbreak (set hardware PC break) 10-116
thbreak (set temporary hardware PC break) 10-116
watch (set data write break) 10-119
rwatch (set data read break) 10-119
awatch (set data read/write break) 10-119
delete (clear break by break number) 10-121
clear (clear break by break position) 10-123
enable (enable breakpoint)disable (disable breakpoint) 10-125
ignore (disable breakpoint with ignore counts) 10-127
info breakpoints (display breakpoint list) 10-128
c33 oab (set on-chip area break) 10-129
c33 obb (set on-chip bus break) 10-131
c33 timebrk (set lapse of time break) 10-135

c33 savebreak (save set breakpoints) 10-136
c33 loadbreak (reload saved breakpoints)..10-137
10.7.9 Symbol Information Display Commands..10-138
info locals (display local symbol)...10-138
info var (display global symbol)..10-138
print (alter symbol value)..10-139
10.7.10 File Loading Commands...10-140
file (load debugging information)...10-140
load (load program)..10-141
10.7.11 Map Information Commands..10-142
c33 rpf (set map information)..10-142
c33 map (display map information)...10-143
10.7.12 Flash Memory Manipulation Commands 10-144
c33 fls (set flash memory)...10-144
c33 fle (erase flash memory) 10-145
10.7.13 Trace Commands...10-146
c33 tm (set PC trace mode)...10-146
c33 td (display PC trace content)..10-152
c33 ts (search PC trace content) 10-153
c33 tf (save PC trace content) 10-154
c33 obt (set on-chip bus trace) 10-155
c33 otd (display on-chip bus trace content) 10-158
c33 otf (save on-chip bus trace content)...10-160
10.7.14 Simulated I/O Commands..10-161
c33 stdin (data input simulation) 10-161
c33 stdout (data output simulation)...10-162
10.7.15 Flash Writer Commands 10-163
c33 fwe (erase program/data)...10-163
c33 fwlp (load program) 10-164
c33 fwld (load data) 10-165
c33 fwdc (copy target memory)..10-166
c33 fwd (display flash writer information)...10-167
10.7.16 Other Commands 10-168
c33 log (logging) 10-168
source (execute command file)...10-169
c33 clockmd (set execution counter mode)...10-170
c33 clock (display execution counter) 10-170
target (connect target) 10-172
detach (disconnect target) 10-174
c33 das (debug unit address set)...10-175
c33 lpt (parallel program load) 10-176
pwd (display current directory)...10-177
cd (change current directory)...10-177
c33 firmupdate (update firmware)...10-178
c33 dclk (change DCLK cycle)...10-179
c33 oscwait (set wait counter in OSC1 mode) 10-180
c33 cachehit (display cache-hit rate) 10-181
c33 logiana (set logic analyzer mode) 10-182
c33 help (help)...10-183
c33 savewatch (saves symbols registered in Watch Window) 10-185
c33 loadwatch (reload saved symbols into Watch Window).....................10-186
quit (quit debugger)...10-187
10.8 Parameter Files...10-188
10.9 Status and Error Messages...10-191
10.9.1 Status Messages 10-191
10.9.2 Error Messages 10-191

11 Other Tools 11-1
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 11-6
11.1.6 Dependency List 11-7
11.1.7 Suffix Definitions 11-10
11.1.8 clean 11-12
11.1.9 Invocation by sh.exe 11-12
11.1.10 Messages 11-13
11.1.11 Precautions 11-13
11.2 ccap.exe 11-14
11.2.1 Function 11-14
11.2.2 Output File 11-14
11.2.3 Method for Using ccap 11-14
11.2.4 Error Messages 11-15
11.3 objdump.exe 11-16
11.3.1 Function 11-16
11.3.2 Input Files 11-16
11.3.3 Method for Using objdump 11-16
11.3.4 Dump Format 11-17
11.3.5 Error Message 11-20
11.3.6 Precautions 11-20
11.4 objcopy.exe 11-21
11.4.1 Function 11-21
11.4.2 Input/Output Files 11-21
11.4.3 Method for Using objcopy 11-22
11.4.4 Creating HEX Files 11-22
11.5 ar.exe 11-23
11.5.1 Function 11-23
11.5.2 Input/Output Files 11-23
11.5.3 Method for Using ar 11-24
11.6 moto2ff.exe 11-26
11.6.1 Function 11-26
11.6.2 Input/Output Files 11-26
11.6.3 Startup Format 11-26
11.6.4 Error/Warning Messages 11-27
11.6.5 Creating Mask ROM Data 11-27

Quick Reference

1 General

1.1 Features

The S1C33 Family C/C++ Compiler Package contains software development tools for compiling C/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 S1C33 Family.
Its principal features are as follows:

Powerful optimizing function

The C/C++ Compiler is designed to suit to the S1C33 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/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/C++ and assembly source level debugger with a simulator function

The debugger supports C/C++ source level debugging and assembly source level debugging. By using the In-Circuit Debugger (S5U1C33000H, S5U1C33001H), 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 Microsoft Windows 2000 Professional, Windows XP, and Windows Vista, the GNU33 IDE is a seamless integrated development environment suitable for a wide range of development tasks, from source creation to debugging.

1.2 Tool Composition

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

(1) C/C++ Compiler (xgcc.exe)

This tool is made based on GNU C/C++ Compiler and is compatible with ANSI C/C++. This tool invokes **cpp.exe** and **cc1.exe/cc1plus.exe** sequentially to compile C/C++ source files to the assembly source files for the S1C33 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/C++ compiler and converts the mnemonics of the source files into object codes (machine language) of the S1C33000. 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/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 the hardware tool (S5U1C33001H or S5U1C33000H) or the debug monitor (S5U1C330M2S). 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/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) GNU33 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 "readmeV3*x*.txt" 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 S1C33 Family C/C++ Compiler Package and their installation methods.

2.1 Working Environment

To use the S1C33 Family C/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 Pentium3 800 MHz, and 512MB or more of memory is recommended.
To use the optional In-Circuit Debugger S5U1C33000H or Debug Monitor S5U1C330M2S with the S5U1C-330M1D1 board, the personal computer also requires a serial port (with a D-sub 9 pin). When using the optional In-Circuit Debugger S5U1C33001H, 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.

Hard drive

The hard drive must have at least 500MB of empty space to install the S1C33 Family C/C++ Compiler Package.

Mouse

A mouse is necessary to operate the tools.

Debugging tool

To debug the program and the target system, an optional In-Circuit Debugger (S5U1C33000H or S5U1C33001H), or Debug Monitor (S5U1C330M2S and S5U1C330M1D1) is needed in addition to this software package.

System software

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

User account

Run the S1C33 Family C Complier Package with Administrator privileges.

Other

- Please go through the precautions and restrictions given in "readmeV3*x*.txt" (English, Japanese) (3*x* indicates version) on the disk.
- Running the tools in this package presumes the presence of **cygwin1.dll**. Although **cygwin1.dll** is stored in the \gnu33 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 \gnu33 directory.

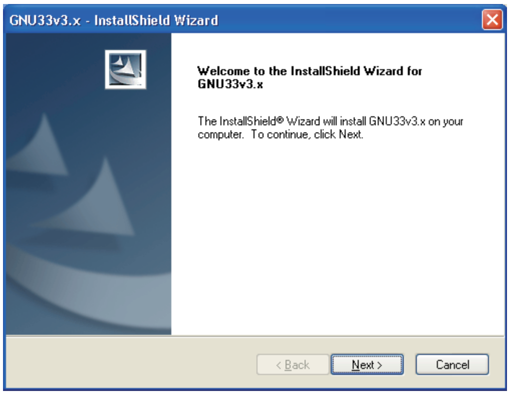
2.2 Installation Method

Installing the tools

(1) Start Windows 2000/XP/Vista.
If Windows is already running, close all other programs that are currently open.

(2) Download the S5U1C33001C archive file from the SEIKO EPSON user's site and extract it into a folder.

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

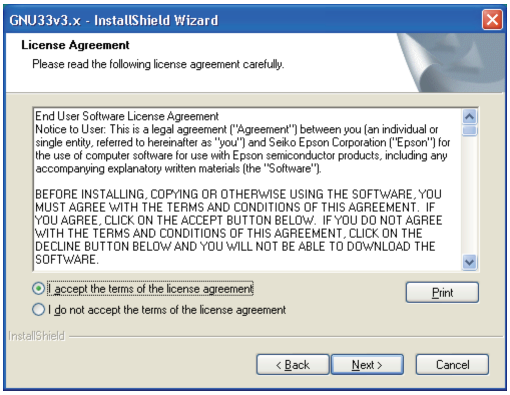


Welcome to the InstallShield Wizard for GNU33v3.x

The InstallShield® Wizard will install GNU33v3.x on your computer. To continue, click Next.

You will see the install wizard start screen.

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



License Agreement

Please read the following license agreement carefully.

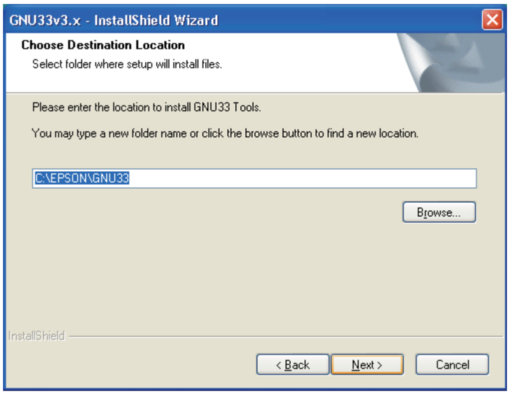
End User Software License Agreement
Notice to User: This is a legal agreement ("Agreement") between you (an individual or single entity, referred to hereinafter as "you") and Seiko Epson Corporation ("Epson") for the use of computer software for use with Epson semiconductor products, including any accompanying explanatory written materials (the "Software").

BEFORE INSTALLING, COPYING OR OTHERWISE USING THE SOFTWARE, YOU MUST AGREE WITH THE TERMS AND CONDITIONS OF THIS AGREEMENT. IF YOU AGREE, CLICK ON THE ACCEPT BUTTON BELOW. IF YOU DO NOT AGREE WITH THE TERMS AND CONDITIONS OF THIS AGREEMENT, CLICK ON THE DECLINE BUTTON BELOW AND YOU WILL NOT BE ABLE TO DOWNLOAD THE SOFTWARE.

● I accept the terms of the license agreement
○ I do not accept the terms of the license agreement

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.



Choose Destination Location

Select folder where setup will install files.

Please enter the location to install GNU33 Tools.

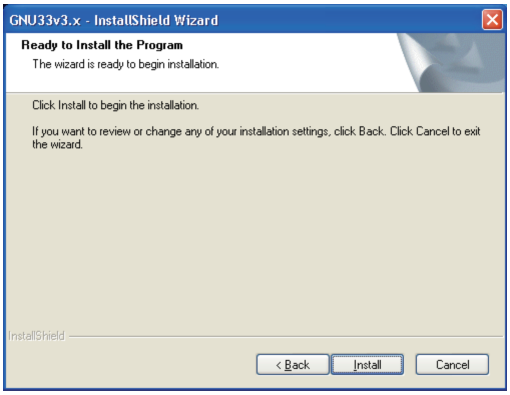
You may type a new folder name or click the browse button to find a new location.

C:\EPSON\GNU33

The screen displayed allows you to select the directory into which the gnu33 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.
If you specify the directory in which an old version has been installed, you are prompted to choose either uninstalling it or changing the install directory by a warning message displayed. The old version may be left on the disk by specifying another directory.

(7) Click the [Next >] button.



GNU33v3.x - InstallShield Wizard

Ready to Install the Program

The wizard is ready to begin installation.

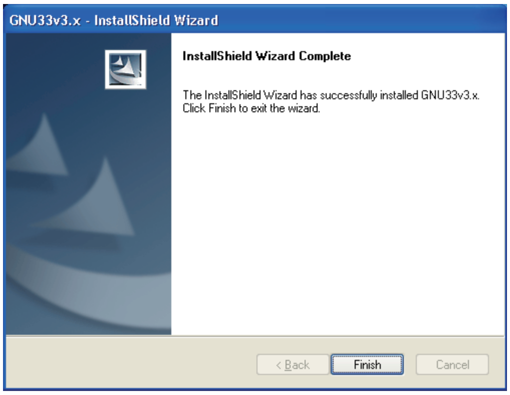
Click Install to begin the installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

InstallShield
< Back
Install
Cancel

This is the install start screen.

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



GNU33v3.x - InstallShield Wizard

InstallShield Wizard Complete

The InstallShield Wizard has successfully installed GNU33v3.x. Click Finish to exit the wizard.

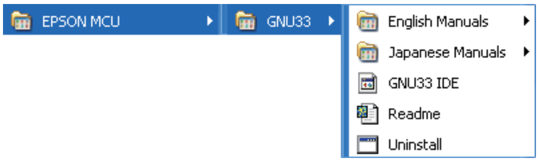
< Back
Finish
Cancel

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] > [GNU33] menu will be added to the Windows startup menu.



EPSON MCU
GNU33
English Manuals
Japanese Manuals
GNU33 IDE
Readme
Uninstall

Installed files

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

\EPSON (default)	
\gnu33 (Root DIR of the gnu33 tool)	
readmeV3*x*.txt	Information about tools (in English and Japanese) with 3*x* indicating version.
Copying.GNU	GNU copyright
xgcc.exe, cpp.exe, cc1.exe/cc1plus.exe	C/C++ compiler
xgcc_filt.exe	Kanji filter (same as xgcc.exe)
as.exe	Assembler filter
as_org.exe	Assembler
ld.exe	Linker
ar.exe	Librarian
gdb.exe	Debugger
make.exe	make
objdump.exe	Object file information display utility
objcopy.exe	Object file copy and translate utility
moto2ff.exe	Mask ROM file generation (ff filling) tool
rm.exe	File remove utility
sed.exe	Stream editor
cp.exe	File copy utility
sh.exe	Bourn shell utility
ccap.exe	Console capture utility
gdbtk.ini	Debugger setup file
resetcold.gdb	Debugger command file for the [Reset cold] button
resethot.gdb	Debugger command file for the [Reset hot] button
userdefine.gdb	Debugger command file for the [User Command] button
savebreak.gdb	Command file for commands related to saving breakpoints
loadbreak.gdb	Command file for commands related to resetting breakpoints
gnuEdit.gdb	Command file for saving external editor name and parameters
cygitcl30.dll, cygitk30.dll, cygtcl80.dll, cygtk80.dll, tix4180.dll	dll files for debugger
cygwin1.dll	dll file for development tools
cygiconv-2.dll	dll for cygwin1.dll
cygintl-3.dll	dll for cygwin1.dll
cygintl-8.dll	dll for cygwin1.dll
\eclipse	
eclipse.exe	GNU33 IDE executable file
eclipse.ini	Eclipse settings file
.eclipseproduct	Eclipse version information
epl-v10.html	EPL license
notice.html	Software agreement
gnu33_32_trans.ico	gnu33 icon file
artifacts.xml	Eclipse update manager file
\configuration	Startup configuration file and other files
\cpuinfo	CPU file folder for wizard
\dropins	plug-in folder (empty folder)
\features	Features
\jre	Java virtual machine
\plugins	Plug-in
\readme	Release note
\lib	Library files
\std	Library for C33 STD Core
libc.a	ANSI C library
libgcc.a	Emulation library
libgccP.a	High-accuracy emulation library
libgcc2.a	long long emulation library
libstdc++.a	C++ library

libstdio.a	Simulated I/O library
\pe	Library for C33 PE Core (configured in the same way as std)
\33401	Library for C33 ADV Core (configured in the same way as std)
\include	C/C++ include files
\sample_ide	Sample files for GNU33 IDE
\utility	Utilities
\sample_std	Sample files for C33 STD Core
\sample_pe	Sample files for C33 PE Core
\sample_adv	Sample files for S1C33401
\sample_c++	C++ sample files
\doc	Manual and other documents
\tool (Middleware directory)	
\tps33g	Toppers directory (used in tutorial)

Refer to the "readmeV3*x*.txt" for the contents of the "sample_ide" and "utility" directories.

Old sample programs

The old sample programs (\sample_std, \sample_pe, \sample_adv, and \sample_c++) have been moved to the \utility directory. To use the samples, move these folders to the \gnu33 directory.

Precaution when installing over existing version

If the gnu33 tools have been installed over the old version, changes in the **GNU33 IDE** may not be reflected in the new version just installed. If this happens, temporarily close **GNU33 IDE**, then start from the command line prompt, as described below.

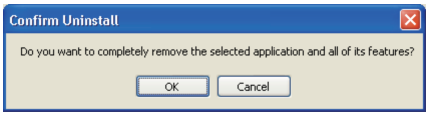
```
C:\EPSON\gnu33\eclipse>eclipse.exe -clean
```

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 COM or USB port for the debugging tool (S5U1C33000H, S5U1C33001H or S5U1C330M2S) in other drivers and applications. Furthermore, make sure that the port has been enabled when using a note PC as some can disable COM and USB ports.
- If the **gdb** debugger or **GNU33 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.

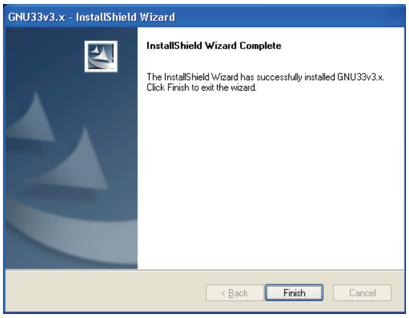
Uninstalling the tools

To uninstall the tools, select [UnInstall] from [EPSON MCU] > [GNU33] in Windows startup menu, then click the [OK] button in the subsequent dialog box.



Do you want to completely remove the selected application and all of its features?

→



InstallShield Wizard Complete

The InstallShield Wizard has successfully installed GNU33v3.x.
Click Finish to exit the wizard.

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

Note: If you set the \EPSON\gnu33\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/C++ compiler tools in this package is made based on the GNU C/C++ Compiler designed by Free Software Foundation, Inc. Please read the "Copying.GNU" text file for the license before using.

EPL

GNU33 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 \gnu33\eclipse directory.

3 Software Development Procedures

3.1 Software Development Flow

Figure 3.1.1 shows typical software development flow.

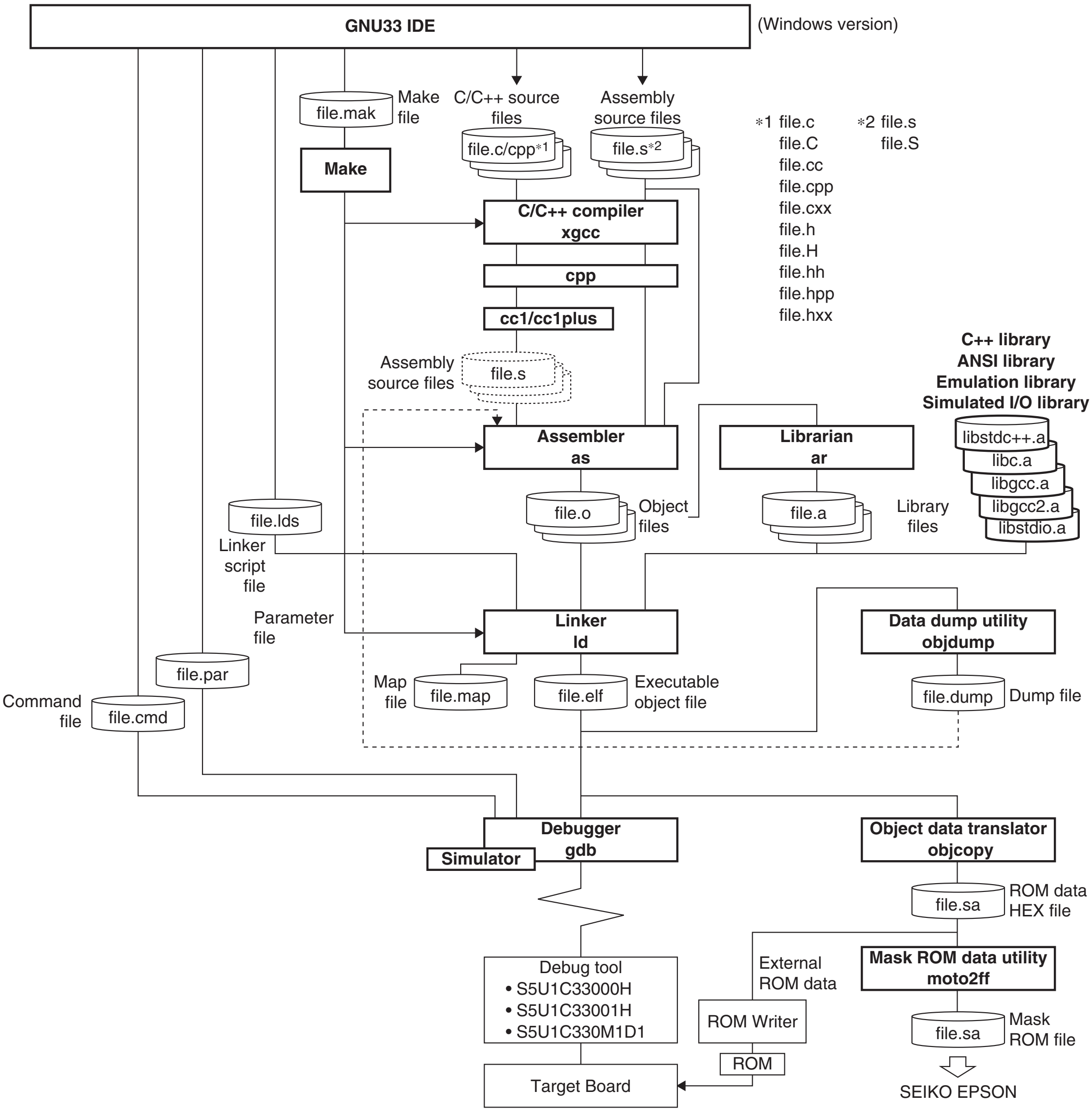

GNU33 IDE
(Windows version)
file.mak
Make file
Make
C/C++ source files
file.c/cpp*1
Assembly source files
file.s*2
*1 file.c
file.C
file.cc
file.cpp
file.cxx
file.h
file.H
file.hh
file.hpp
file.hxx
*2 file.s
file.S
C/C++ compiler
xgcc
cpp
cc1/cc1plus
Assembly source files
file.s
C++ library
ANSI library
Emulation library
Simulated I/O library
Assembler
as
Librarian
ar
libstdc++.a
libc.a
libgcc.a
libgcc2.a
libstdio.a
file.lds
Linker script file
file.o
Object files
file.a
Library files
Parameter file
Linker
ld
Data dump utility
objdump
file.par
Command file
file.cmd
Map file
file.map
file.elf
Executable object file
file.dump
Dump file
Debugger
gdb
Simulator
Object data translator
objcopy
file.sa
ROM data HEX file
External ROM data
Mask ROM data utility
moto2ff
Debug tool
• S5U1C33000H
• S5U1C33001H
• S5U1C330M1D1
ROM Writer
ROM
file.sa
Mask ROM file
Target Board
SEIKO EPSON

Figure 3.1.1 Software development flow

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 **GNU33 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/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 or C++ sources)

  The source files are compiled by the **xgcc** C/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 the S5U1C33000H, S5U1C33001H, and debug monitor 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 S1C33000 Core CPU 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. Finally, convert the HEX file for internal ROM into a mask data file by **moto2ff** and present the converted file to Seiko Epson.

In addition to the tools described above, the C/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 S1C33 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 S1C33 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.

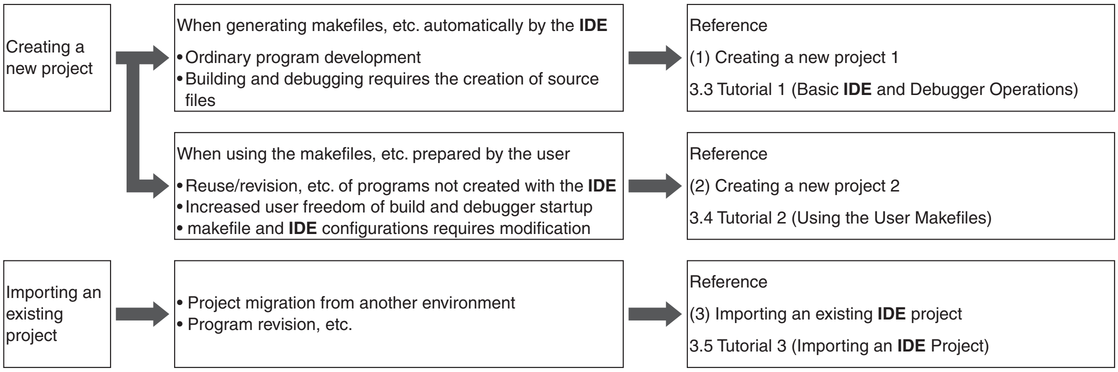

Creating a new project
When generating makefiles, etc. automatically by the **IDE**
• Ordinary program development
• Building and debugging requires the creation of source files
Reference
(1) Creating a new project 1
3.3 Tutorial 1 (Basic **IDE** and Debugger Operations)
When using the makefiles, etc. prepared by the user
• Reuse/revision, etc. of programs not created with the **IDE**
• Increased user freedom of build and debugger startup
• makefile and **IDE** configurations requires modification
Reference
(2) Creating a new project 2
3.4 Tutorial 2 (Using the User Makefiles)
Importing an existing project
• Project migration from another environment
• Program revision, etc.
Reference
(3) Importing an existing **IDE** project
3.5 Tutorial 3 (Importing an **IDE** Project)

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.

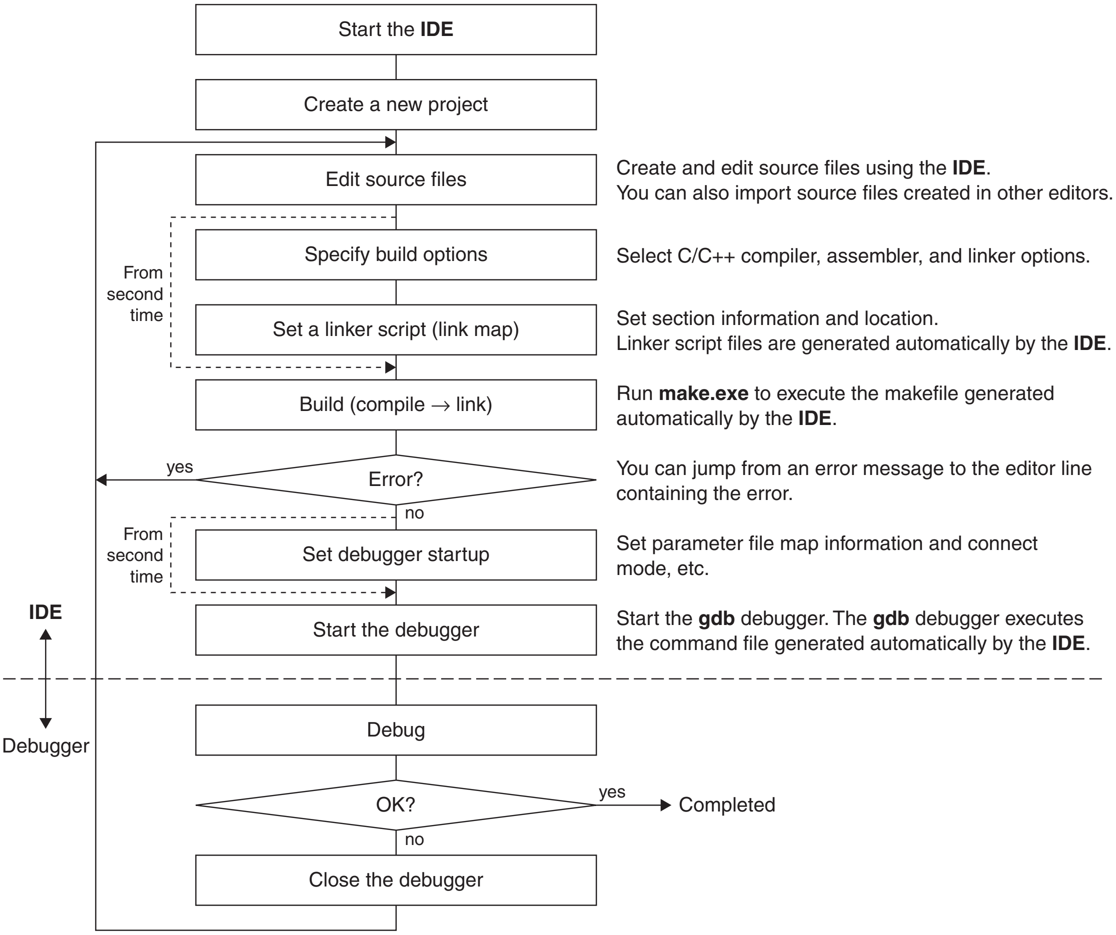

Start the **IDE**
Create a new project
Edit source files
Create and edit source files using the **IDE**.
You can also import source files created in other editors.
Specify build options
Select C/C++ compiler, assembler, and linker options.
From second time
Set a linker script (link map)
Set section information and location.
Linker script files are generated automatically by the **IDE**.
Build (compile → link)
Run **make.exe** to execute the makefile generated automatically by the **IDE**.
yes
Error?
no
You can jump from an error message to the editor line containing the error.
From second time
Set debugger startup
Set parameter file map information and connect mode, etc.
IDE
Start the debugger
Start the **gdb** debugger. The **gdb** debugger executes the command file generated automatically by the **IDE**.
Debugger
Debug
OK?
yes
Completed
no
Close the debugger

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 revising programs created with the old gnu33 tools or developing new software using such resources, if the makefiles or debug command files created thereby must be used, those files may be used instead of generating files automatically with the **IDE**. The same applies when exclusive user makefiles or debug command files are required to develop new software.

The basic procedural flow is given below.

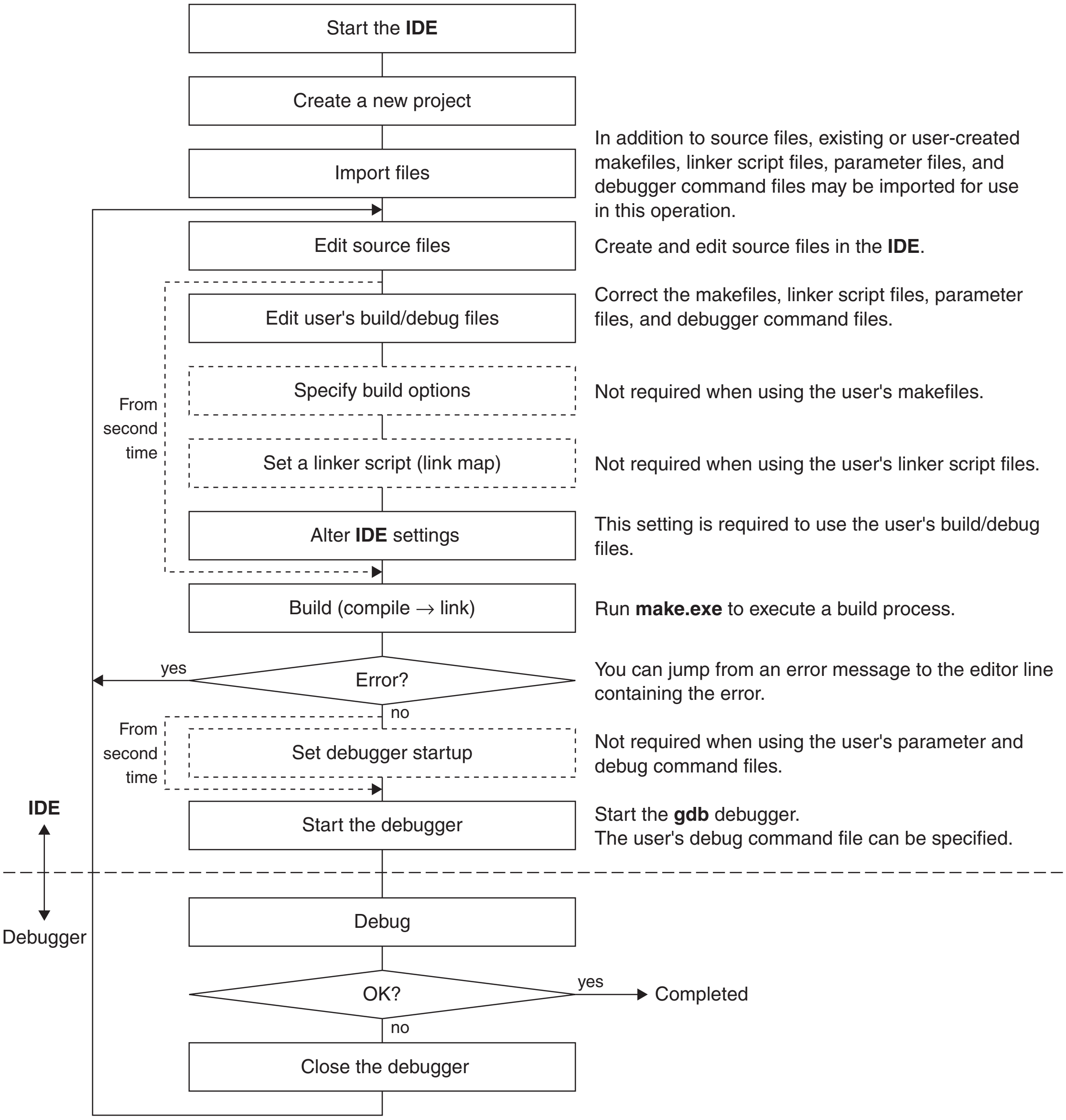

Start the **IDE**
Create a new project
Import files
In addition to source files, existing or user-created makefiles, linker script files, parameter files, and debugger command files may be imported for use in this operation.
Edit source files
Create and edit source files in the **IDE**.
Edit user's build/debug files
Correct the makefiles, linker script files, parameter files, and debugger command files.
Specify build options
Not required when using the user's makefiles.
From second time
Set a linker script (link map)
Not required when using the user's linker script files.
Alter **IDE** settings
This setting is required to use the user's build/debug files.
Build (compile → link)
Run **make.exe** to execute a build process.
yes
Error?
You can jump from an error message to the editor line containing the error.
no
From second time
Set debugger startup
Not required when using the user's parameter and debug command files.
IDE
Start the debugger
Start the **gdb** debugger.
The user's debug command file can be specified.
Debugger
Debug
OK?
yes
Completed
no
Close the debugger

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 created with the old gnu33 tools using the makefiles created at that time.

To use existing 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.

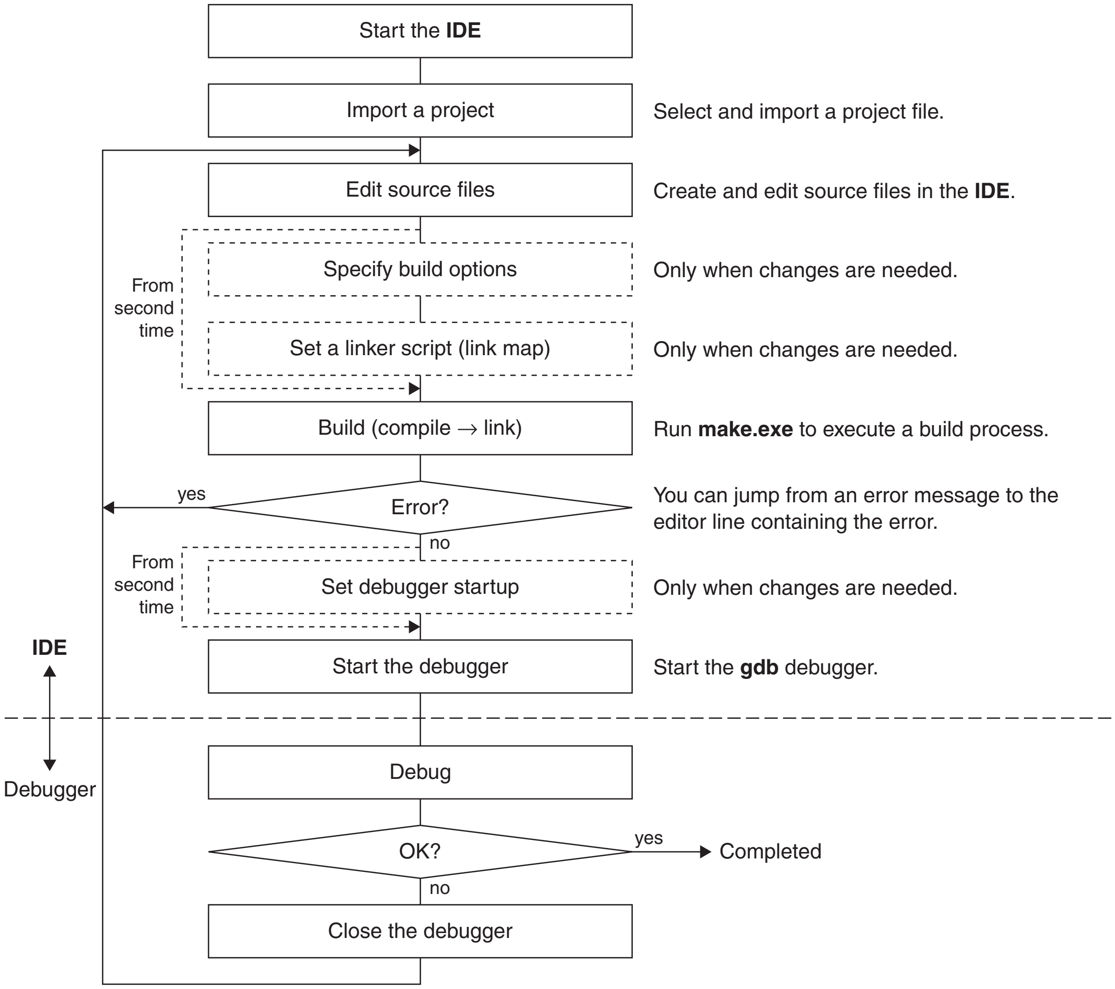

Start the **IDE**
Import a project
Select and import a project file.
Edit source files
Create and edit source files in the **IDE**.
Specify build options
Only when changes are needed.
From second time
Set a linker script (link map)
Only when changes are needed.
Build (compile → link)
Run **make.exe** to execute a build process.
yes
Error?
You can jump from an error message to the editor line containing the error.
no
From second time
Set debugger startup
Only when changes are needed.
IDE
Start the debugger
Start the **gdb** debugger.
Debugger
Debug
OK?
yes
Completed
no
Close the debugger

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\gnu33\sample_ide\std\simulator\tst` directory.

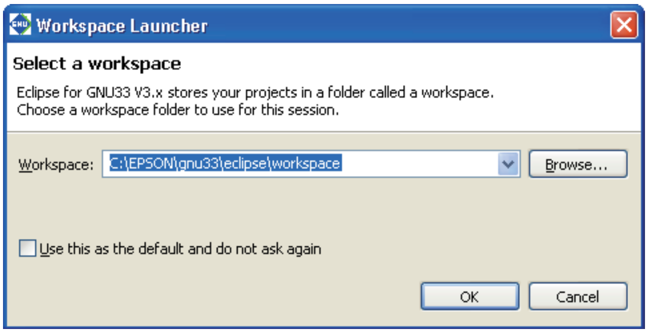
`boot.s`	Assembler source file
`main.c`	C source file

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\gnu33\eclipse` directory to start the **IDE**. You also can start the **IDE** by selecting [EPSON MCU] > [GNU33] > [GNU33 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.


Workspace Launcher

Select a workspace

Eclipse for GNU33 V3.x stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\EPSON\gnu33\eclipse\workspace

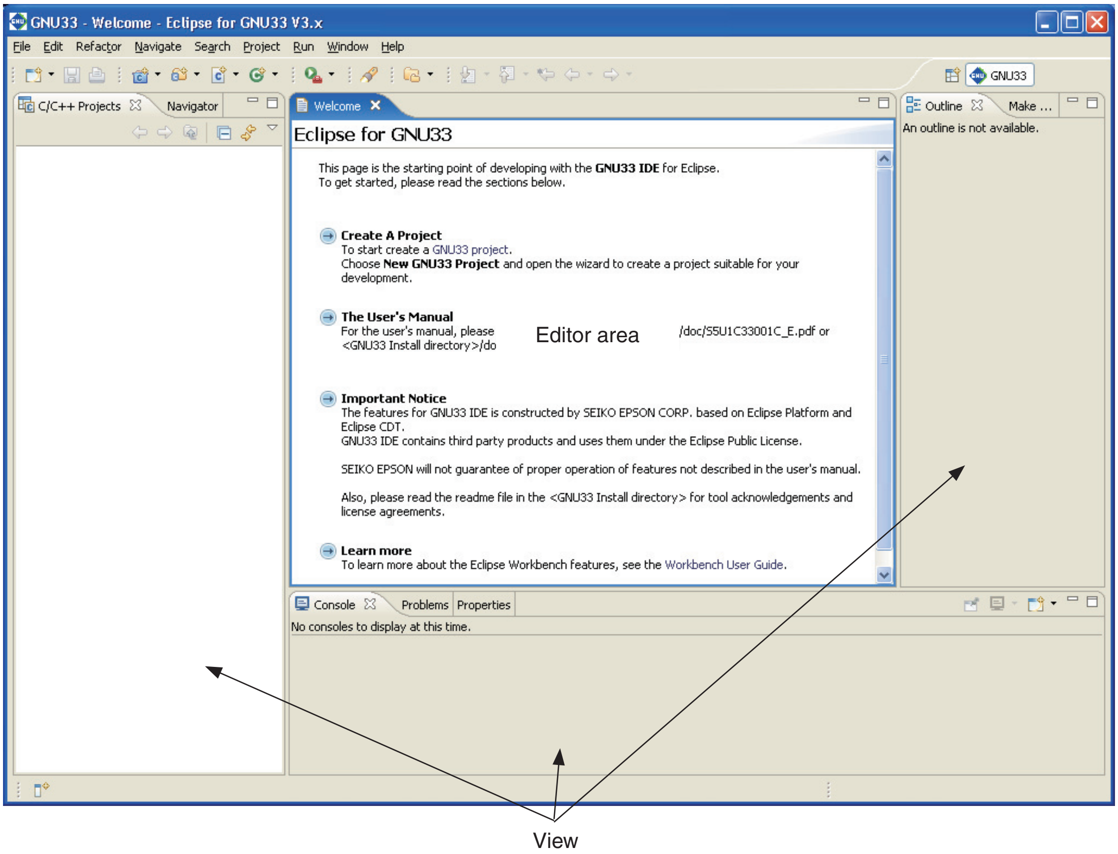
☐ Use this as the default and do not ask again

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

* Do not specify the project directory (directory containing .project file) as a workspace directory. Doing so may result in failures with project imports (when [Copy projects into workspace] is selected).

Step 2: Click the [OK] button.

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


GNU33 - Welcome - Eclipse for GNU33 V3.x
File Edit Refactor Navigate Search Project Run Window Help
GNU33
C/C++ Projects
Navigator
Welcome
Outline
Make ...
Console
Problems
Properties
Editor area
View

Eclipse for GNU33

This page is the starting point of developing with the **GNU33 IDE** for Eclipse.
To get started, please read the sections below.

Create A Project
To start create a GNU33 project.
Choose **New GNU33 Project** and open the wizard to create a project suitable for your development.

The User's Manual
For the user's manual, please /doc/S5U1C33001C_E.pdf or
<GNU33 Install directory>/do

Important Notice
The features for GNU33 IDE is constructed by SEIKO EPSON CORP. based on Eclipse Platform and Eclipse CDT.
GNU33 IDE contains third party products and uses them under the Eclipse Public License.

SEIKO EPSON will not guarantee of proper operation of features not described in the user's manual.

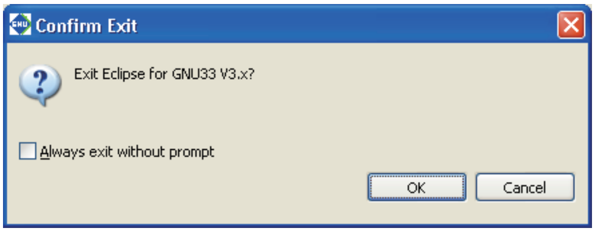
Also, please read the readme file in the <GNU33 Install directory> for tool acknowledgements and license agreements.

Learn more
To learn more about the Eclipse Workbench features, see the Workbench User Guide.

An outline is not available.

No consoles to display at this time.

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.


Confirm Exit
OK
Cancel

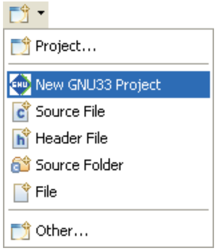
Exit Eclipse for GNU33 V3.x?

☐ Always exit without prompt

3.3.2 Creating a Project

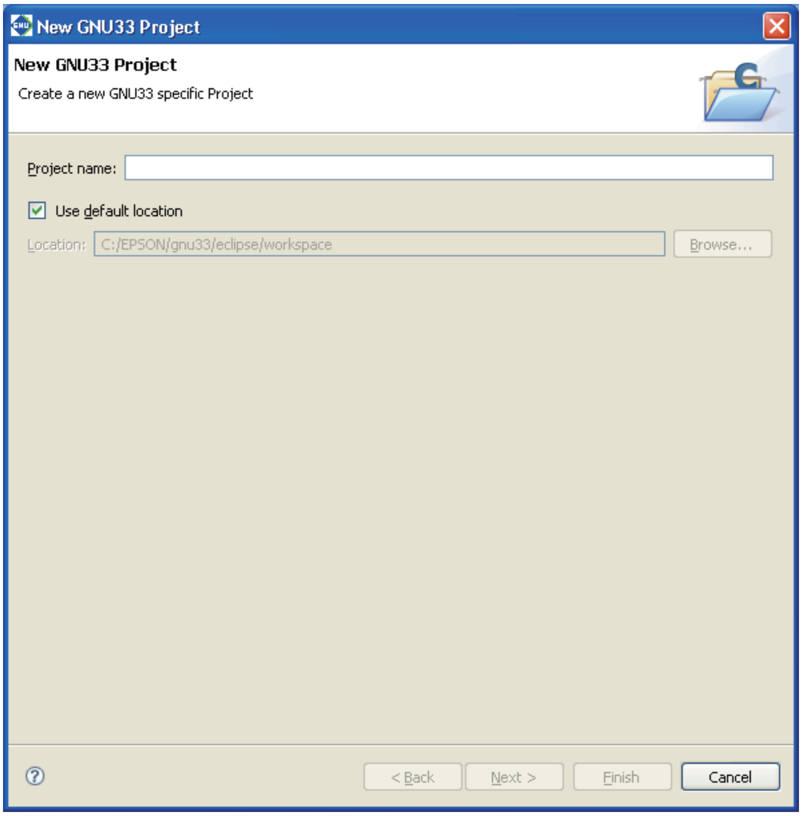
In applications development, a single executable program file is created from multiple source files. To manage these files in one location, you must create a project. The **IDE** generates programs on a per-project basis. In a sense, the project is the application program you want to develop, but the project actually created is a directory with a specified project name, wherein files containing project information (`.cproject`, `.gnu33project`, and `.project`) are generated.

To create a new project


Project...
New GNU33 Project
Source File
Header File
Source Folder
File
Other...

Step 3: Select [New GNU33 Project] from the [New] pulldown menu in the toolbar.

You can also select [New GNU33 Project] from the [File] menu or from [New] on the context menu (displayed by right-clicking) in the [C/C++ Projects/Navigator] view.


New GNU33 Project

New GNU33 Project

Create a new GNU33 specific Project

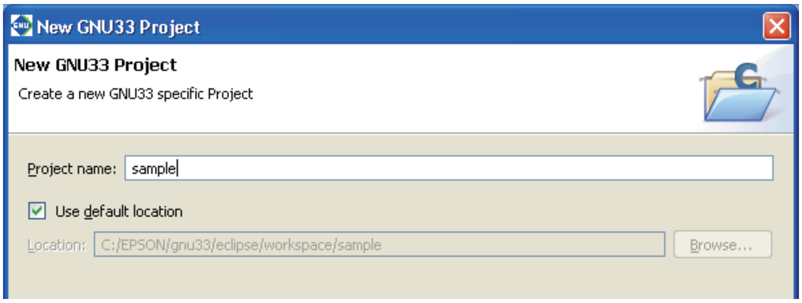
Project name:

☑ Use default location

Location: C:/EPSON/gnu33/eclipse/workspace

Browse...
< Back
Next >
Finish
Cancel

Specifying a project name



New GNU33 Project

Create a new GNU33 specific Project

Project name: sample

☑ Use default location

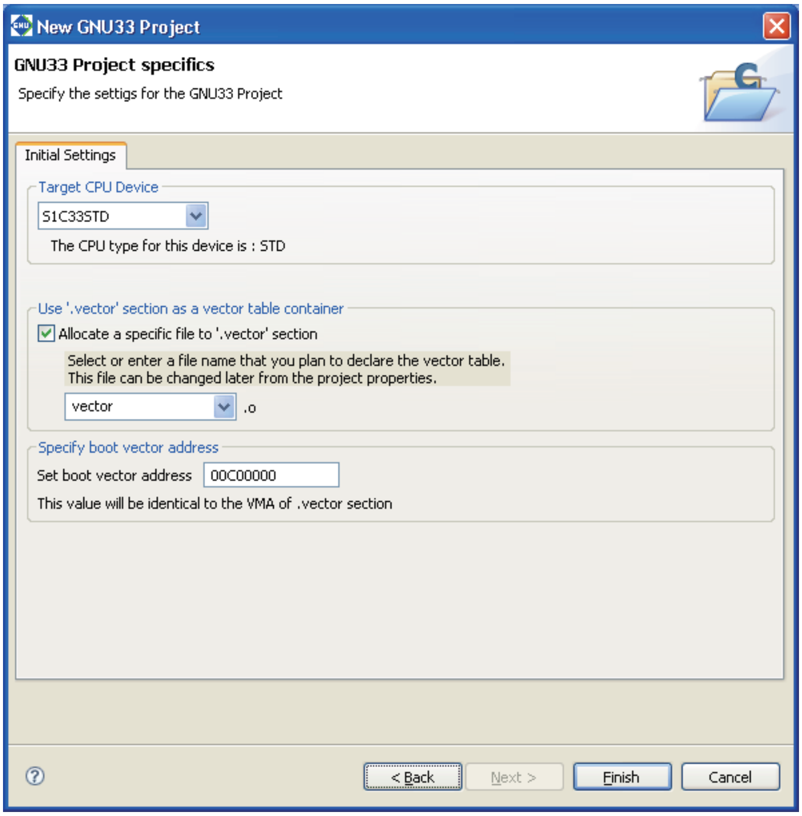
Location: C:/EPSON/gnu33/eclipse/workspace/sample

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

Leave the [Use default] check box in [Project contents] 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 and vector table file.



GNU33 Project specifics

Specify the settigs for the GNU33 Project

Target CPU Device

S1C33STD

The CPU type for this device is : STD

Use '.vector' section as a vector table container

☑ Allocate a specific file to '.vector' section

Select or enter a file name that you plan to declare the vector table.
This file can be changed later from the project properties.

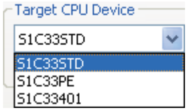
vector .o

Specify boot vector address

Set boot vector address 00C00000

This value will be identical to the VMA of .vector section

Specifying a target CPU

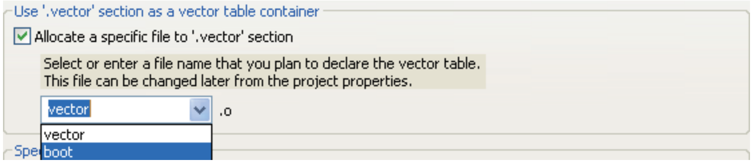


Step 6: From the [Target CPU Device] combo box, select the type of the target processor. Here, select "S1C33STD".

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 interrupt vectors located in memory always begin with the vector 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 7: Select `boot.o` from the pulldown list.



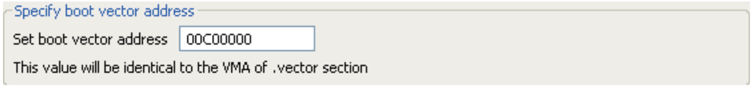
☑ Allocate a specific file to '.vector' section

Select or enter a file name that you plan to declare the vector table.
This file can be changed later from the project properties.

For detailed information on the `.vector` section, refer to Section 5.7.7, "Editing a Linker Script".

Specifying the boot vector address

In the [Specify boot vector address] field, specify a boot vector address. The default value is "00C00000" when "S1C33STD" or "S1C33PE" is selected as the target CPU or "20000000" when "S1C33401" is selected. 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**. It will also 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.

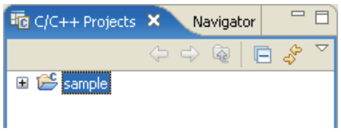


Set boot vector address 00C00000

This value will be identical to the VMA of .vector section

Step 8: Click the [Finish] button.

The [New GNU33 Project] wizard will be closed, creating a project with the specified name.



The target CPU and the vector table file can be revised later.

3.3.3 Creating, Adding, and Editing a Source File

The **IDE** supports C, 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

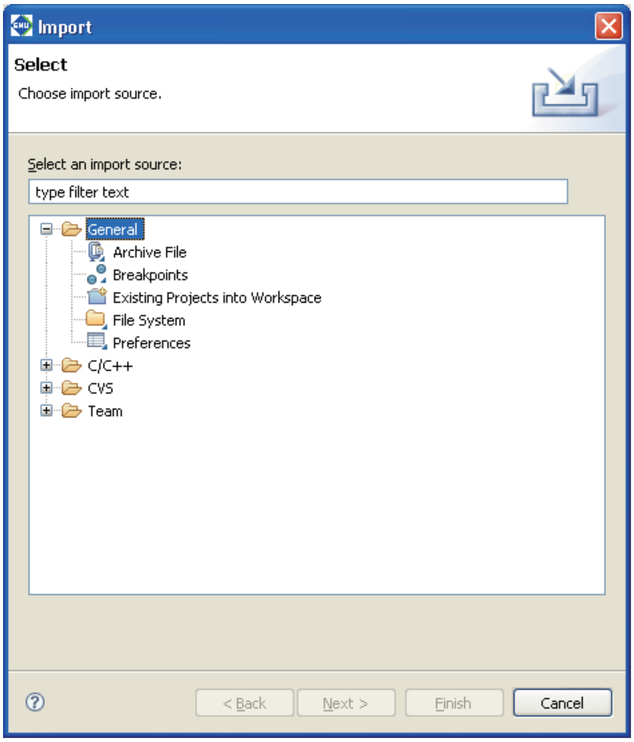
Use the **IDE** editor or a general-purpose editor to create a source file. You can also use an existing source file in the application you created for the S1C33 Family.
In this tutorial, we will use the source files prepared as examples.
For detailed information on creating a new source file with the **IDE**, refer to Section 5.5, "The Editor and Editing Source Files".

Adding a source file

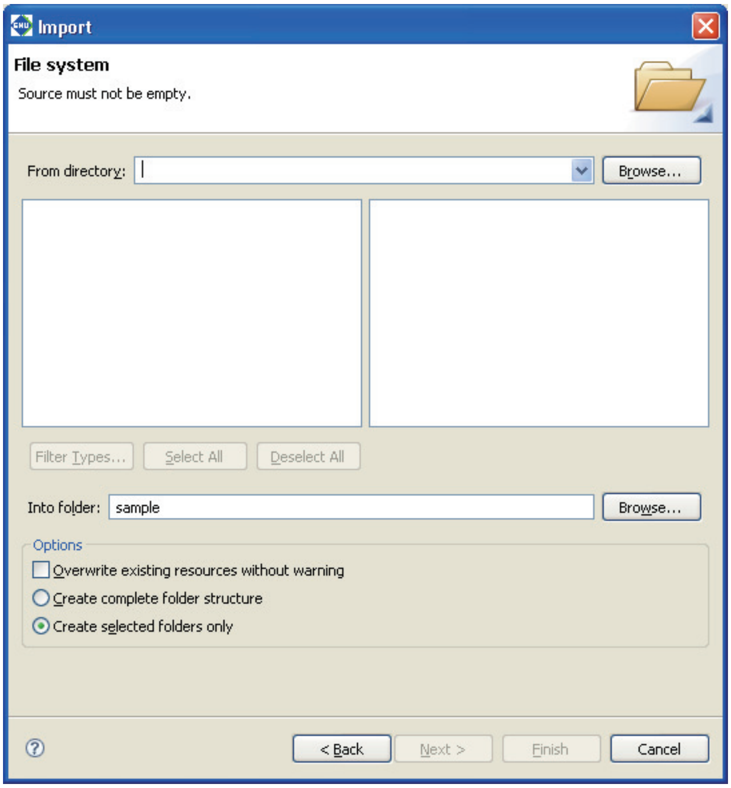
Load the source files prepared as samples into the project.

Step 9: Select [Import...] from the [File] menu.

The [Import] wizard will start.


Import
Select
Choose import source.
Select an import source:
type filter text
General
Archive File
Breakpoints
Existing Projects into Workspace
File System
Preferences
C/C++
CVS
Team
< Back
Next >
Finish
Cancel

Step 10: From the list displayed, select [General] > [File System] and click the [Next>] button.


Import
Filter Types...
Select All
Deselect All
Browse...
Browse...
< Back
Next >
Finish
Cancel

File system

Source must not be empty.

From directory:

Into folder: sample

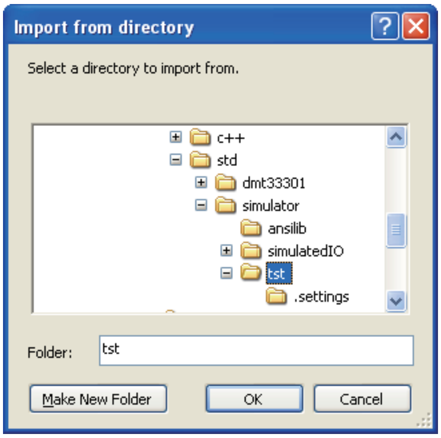
Options

☐ Overwrite existing resources without warning

○ Create complete folder structure

● Create selected folders only

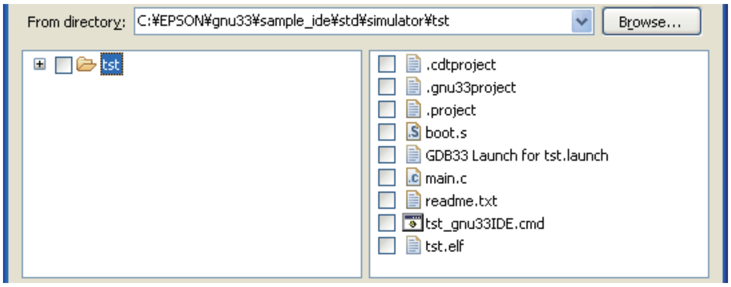
Step 11: Click the [Browse...] button for [From directory:]. The [Import from directory] dialog box will be displayed, so select the `\EPSON\gnu33\sample_ide\std\simulator\tst` directory from the drive (C) in which you installed the **IDE** and click [OK].


Import from directory
c++
std
dmt33301
simulator
ansilib
simulatedIO
tst
.settings
Make New Folder
OK
Cancel

Select a directory to import from.

Folder: tst

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.

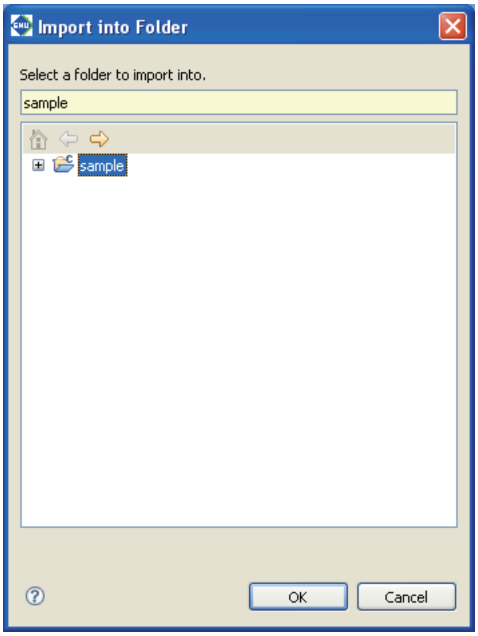


From directory: C:¥EPSON¥gnu33¥sample_ide¥std¥simulator¥tst

☐ .cdtproject
☐ .gnu33project
☐ .project
☐ boot.s
☐ GDB33 Launch for tst.launch
☐ main.c
☐ readme.txt
☐ tst_gnu33IDE.cmd
☐ tst.elf

Step 12: 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).

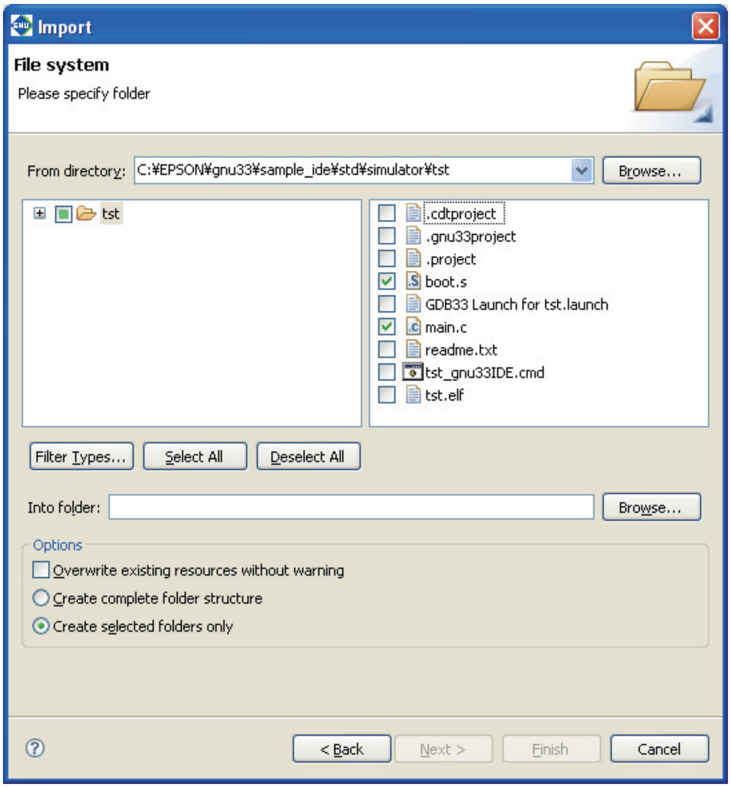
Step 13: Click the [Browse...] button for the [Into folder:]. This displays the [Import into Folder] dialog box. Select the "`sample`" folder and click [OK].


Import into Folder
OK
Cancel

Select a folder to import into.

sample

⊞ sample


Import
Filter Types...
Select All
Deselect All
Browse...
Browse...
< Back
Next >
Finish
Cancel

File system

Please specify folder

From directory: C:¥EPSON¥gnu33¥sample_ide¥std¥simulator¥tst

⊞ ☑ tst

☐ .cdtproject
☐ .gnu33project
☐ .project
☑ boot.s
☐ GDB33 Launch for tst.launch
☑ main.c
☐ readme.txt
☐ tst_gnu33IDE.cmd
☐ tst.elf

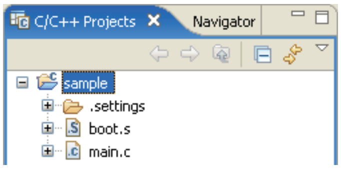
Into folder:

Options
☐ Overwrite existing resources without warning
○ Create complete folder structure
● Create selected folders only

Step 14: After confirming that the dialog box is filled out as shown above, click the [Finish] button.

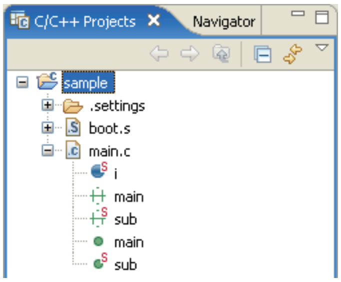
This procedure adds "`boot.s`" and "`main.c`" to the project.

Step 15: Double-click "`sample`" in the [C/C++ Projects] view, or click [+] shown before "`sample`".


C/C++ Projects
Navigator
sample
.settings
boot.s
main.c

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

Step 16: Click [+] for "`main.c`" in the [C/C++ Projects] view.

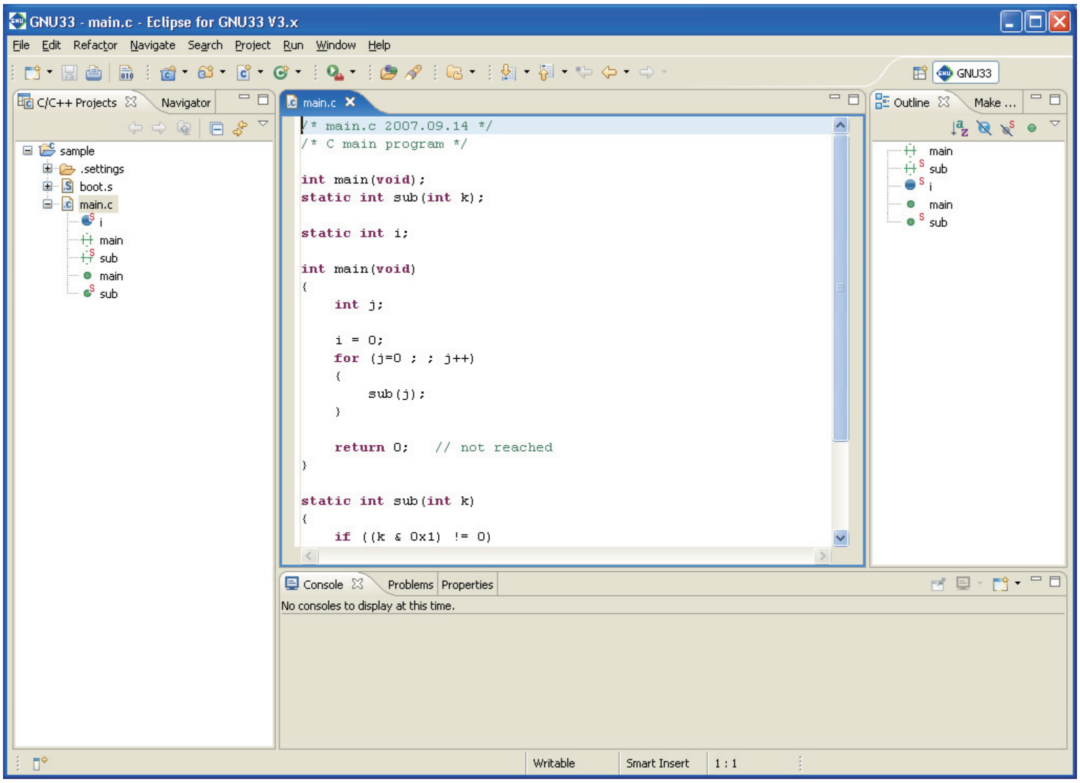

C/C++ Projects
Navigator
sample
.settings
boot.s
main.c
i
main
sub
main
sub

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

Displaying and editing source files

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

Step 17: Double-click "`main.c`" in the [C/C++ Projects] view.


GNU33 - main.c - Eclipse for GNU33 V3.x
File Edit Refactor Navigate Search Project Run Window Help
GNU33
C/C++ Projects
Navigator
sample
.settings
boot.s
main.c
i
main
sub
main
sub
main.c
Outline
Make ...
main
sub
i
main
sub
Console
Problems
Properties
Writable
Smart Insert
1 : 1

```
/* main.c 2007.09.14 */
/* C main program */

int main(void);
static int sub(int k);

static int i;

int main(void)
{
    int j;

    i = 0;
    for (j=0 ; ; j++)
    {
        sub(j);
    }

    return 0;   // not reached
}

static int sub(int k)
{
    if ((k & 0x1) != 0)
```

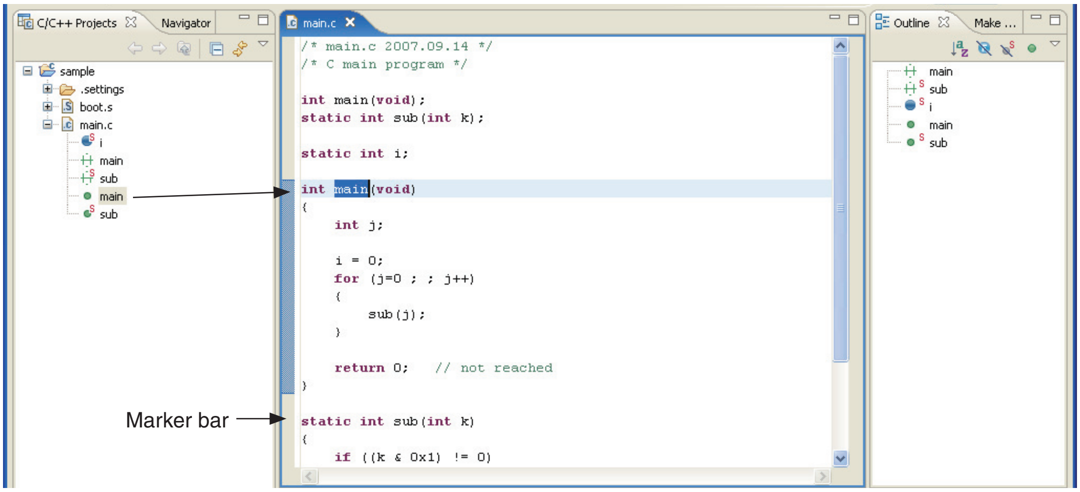
No consoles to display at this time.

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 or C++ sources are displayed, reserved words, comments, and C/C++ strings are highlighted in color.

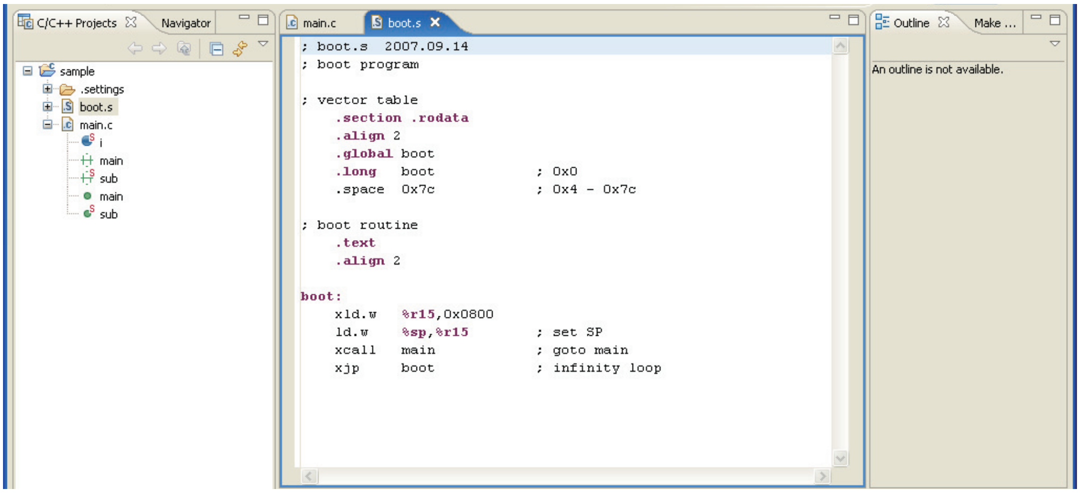
Step 18: Double-click "● `main`" in the [C/C++ Projects] view.


C/C++ Projects
Navigator
sample
.settings
boot.s
main.c
i
main
sub
main
sub
Marker bar
main.c
Outline
Make ...
main
sub
i
main
sub

```
/* main.c 2007.09.14 */
/* C main program */

int main(void);
static int sub(int k);

static int i;

int main(void)
{
    int j;

    i = 0;
    for (j=0 ; ; j++)
    {
        sub(j);
    }

    return 0;    // not reached
}

static int sub(int k)
{
    if ((k & 0x1) != 0)
```

The editor will jump to the line where `int main(void)` 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.
The same effect may be obtained by clicking on "● `main`" in the [Outline] view (right side view).

Step 19: Double-click "`boot.s`" in the [C/C++ Projects] view.


C/C++ Projects
Navigator
sample
.settings
boot.s
main.c
i
main
sub
main
sub
main.c
boot.s
Outline
Make ...

```
; boot.s  2007.09.14
; boot program

; vector table
    .section .rodata
    .align 2
    .global boot
    .long   boot              ; 0x0
    .space  0x7c              ; 0x4 - 0x7c

; boot routine
    .text
    .align 2

boot:
    xld.w   %r15,0x0800
    ld.w    %sp,%r15          ; set SP
    xcall   main              ; goto main
    xjp     boot              ; infinity loop
```

An outline is not available.

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 20: 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

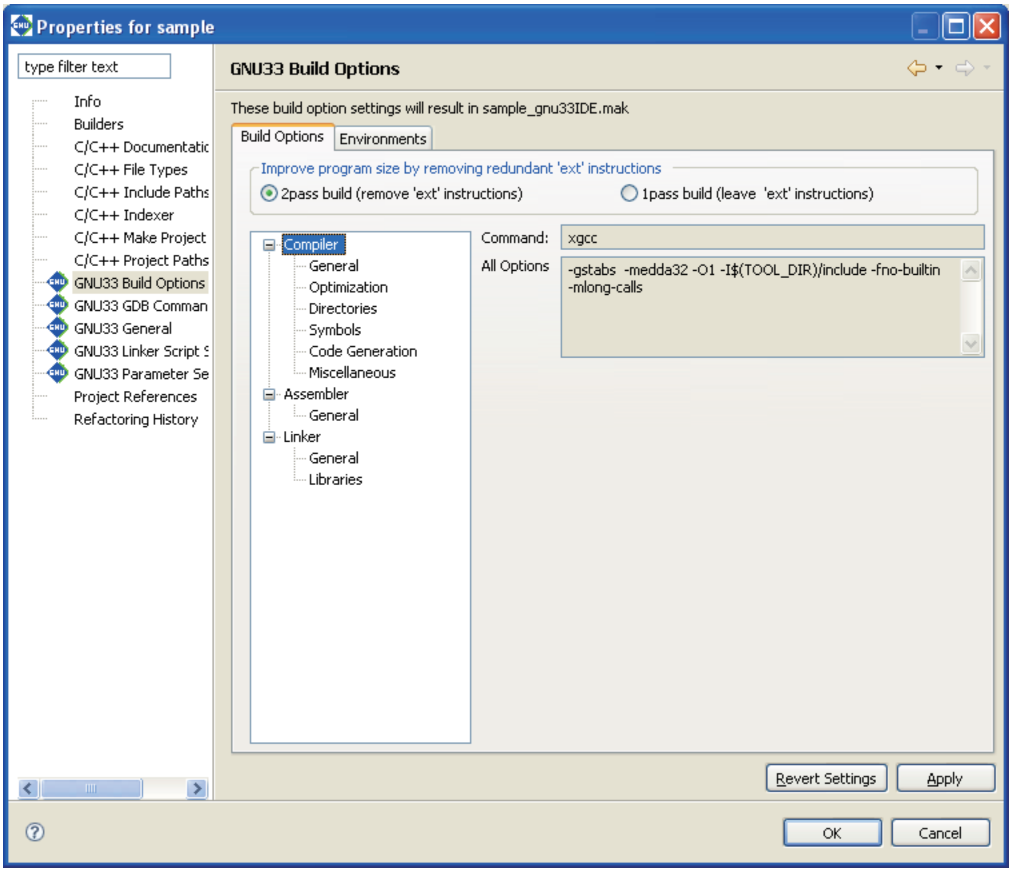
To build a project (to generate an executable object file), **make.exe** is used to start the compiler, assembler, and linker. Although the makefiles required for build are generated automatically by the **IDE**, the build options to be written in those files (i.e., compiler, assembler, and linker startup 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 21: 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 22: From the properties list on the left side of the dialog box, select [GNU33 Build Options] by clicking on it to display the [Build Options] tab page.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

Command: xgcc

All Options: -gstabs -medda32 -O1 -I$(TOOL_DIR)/include -fno-builtin -mlong-calls

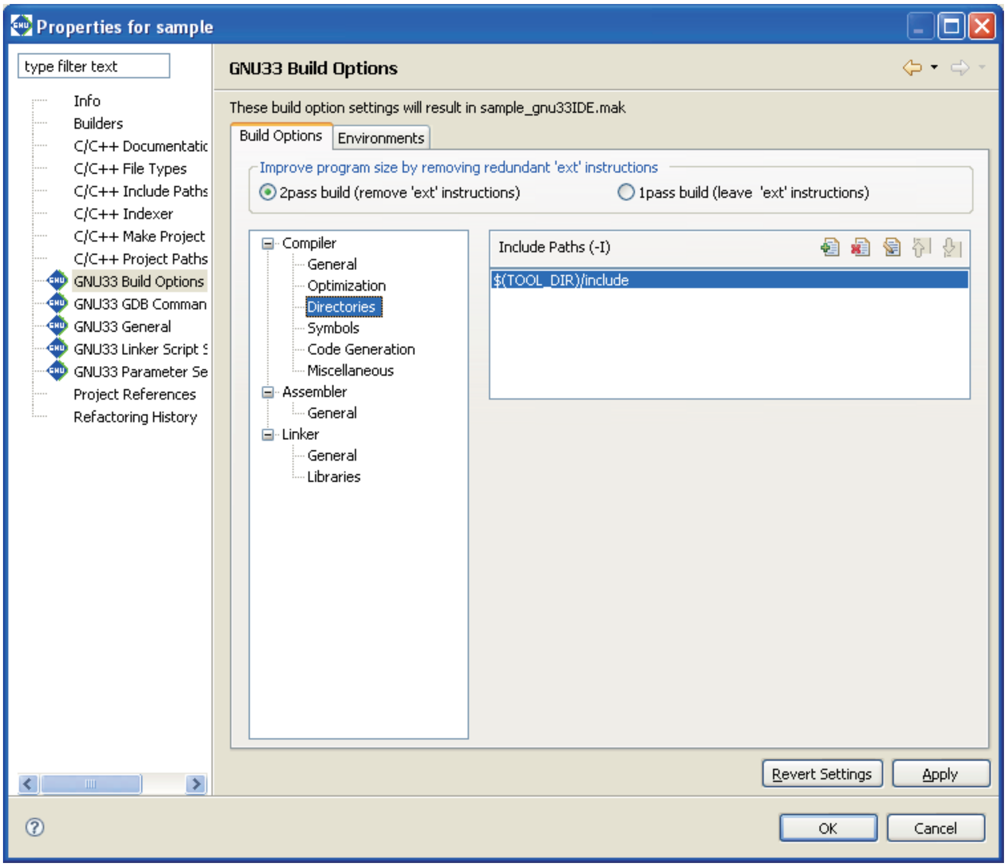
Here, you can set command line options for the compiler, assembler, and linker.
When you select one of the tool names shown in tree form (Compiler, Assembler, or Linker) by clicking on it, the currently selected options are displayed in the [All Options] column. Select the kind of option from those shown 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 23 to 26 below are shown for reference only. No operation is required.

Step 23: Select [Compiler] > [Directories] from the [Build Options] tree.

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


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Improve program size by removing redundant 'ext' instructions
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Include Paths (-I)
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

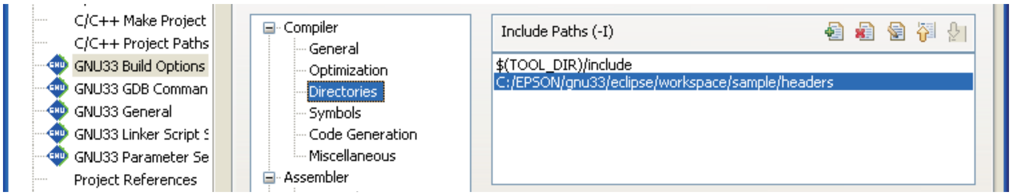
● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

Include Paths (-I): $(TOOL_DIR)/include

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

Step 24: 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.


C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
Include Paths (-I)

$(TOOL_DIR)/include
C:/EPSON/gnu33/eclipse/workspace/sample/headers

The features of other buttons are summarized below.

[Delete]	Deletes the path selected in the list.
[Edit]	Edits the path selected in the list. A dialog box is displayed in which you can modify the path.
[Move Up]	Moves the path selected in the list one place up in the list. Include files are searched sequentially, beginning with the path uppermost in the list.
[Move Down]	Moves the selected path down in the list.

Step 25: Press the [Apply] button to confirm the changes made here.

The directory setting specified here provides the includes file paths for the C/C++ compiler to search from.

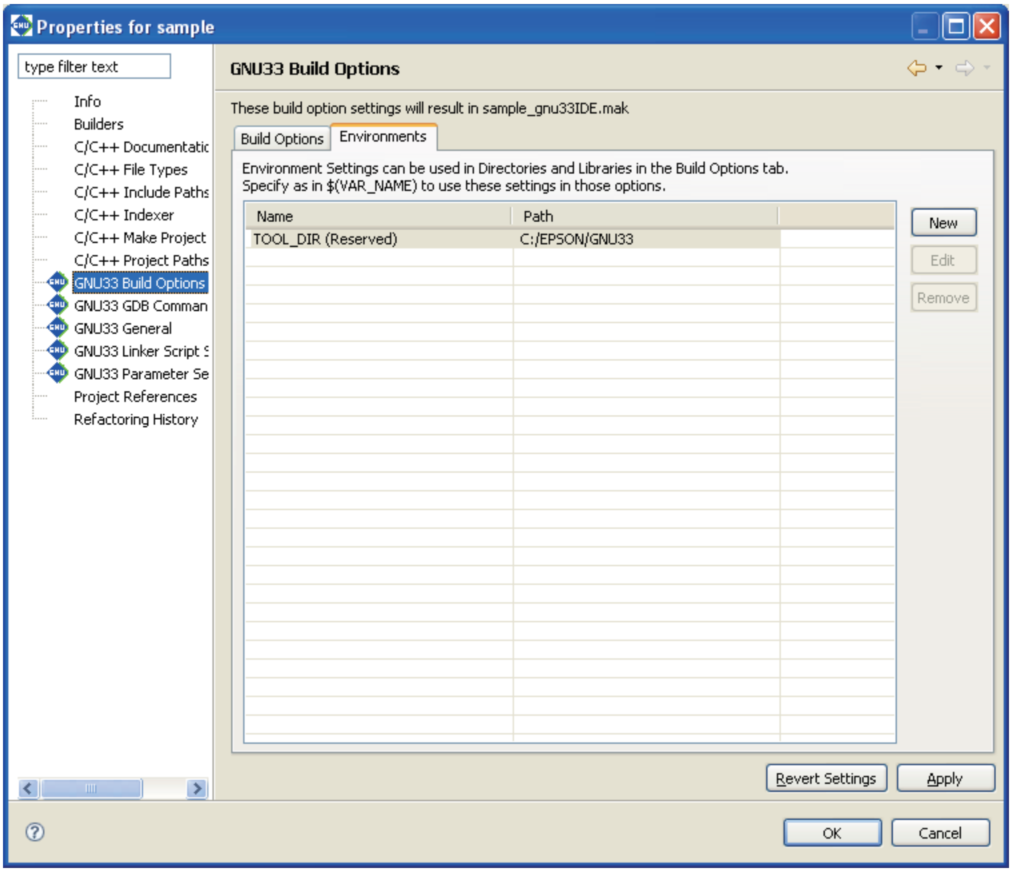
Macro and environment variable to specify a path

The [Include Paths (-I)] column lists "`$(TOOL_DIR)/include`" that is set by default.
`$(`*`environment variable`*`)` is 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 gnu33 tool directory is defined. The defined contents can be verified in the [Environments] tab page.
Example: If the gnu33 tools have been installed in the c:\EPSON\gnu33 directory

```
TOOL_DIR = c:/EPSON/gnu33
```

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/gnu33/include`.

The [Environments] tab page allows the user to define environment variables similar to `TOOL_DIR`.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
New
Edit
Remove
Revert Settings
Apply
OK
Cancel

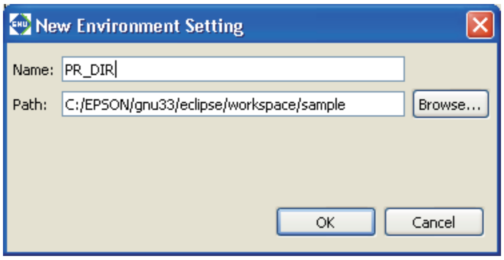
These build option settings will result in sample_gnu33IDE.mak

Environment Settings can be used in Directories and Libraries in the Build Options tab.
Specify as in $(VAR_NAME) to use these settings in those options.

Name	Path
TOOL_DIR (Reserved)	C:/EPSON/GNU33

The definition procedure is described below.

Step 26: 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.


New Environment Setting
Browse...
OK
Cancel

Name: PR_DIR
Path: C:/EPSON/gnu33/eclipse/workspace/sample

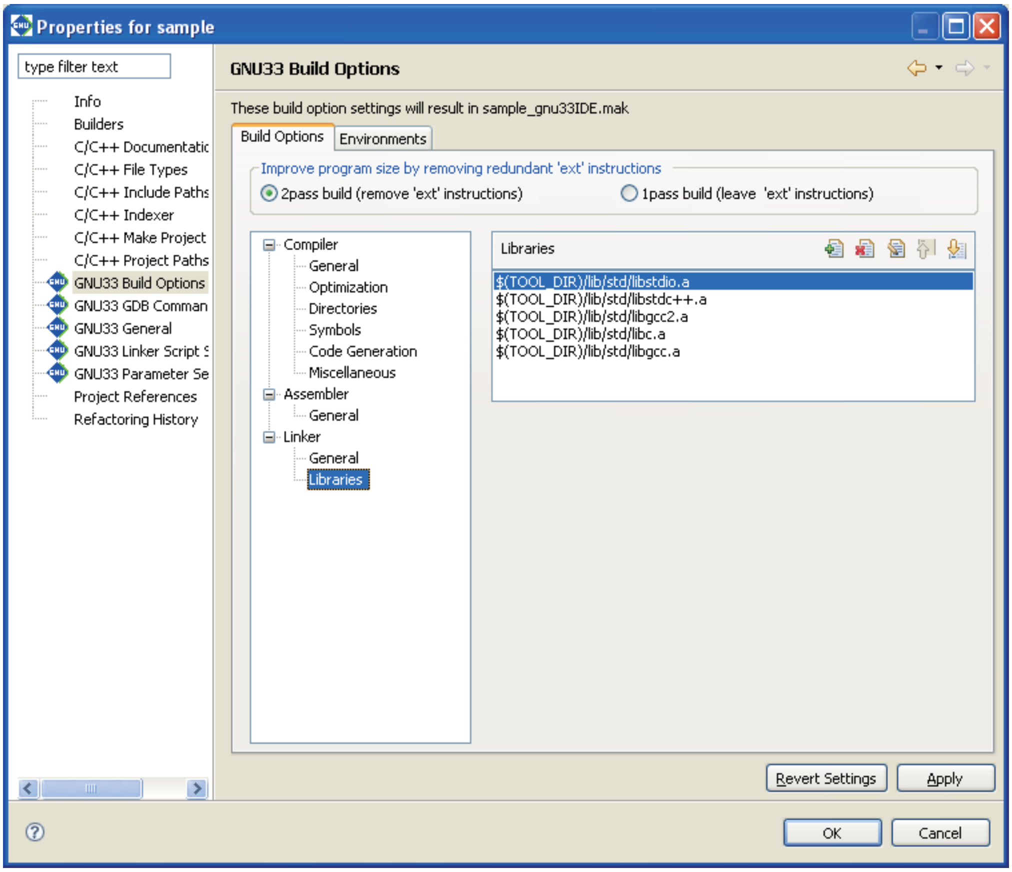
The environment variables defined here may be used for specifying include file and library file paths in the build options. The environment variable should be used as a `$(`*environment variable*`)` macro format when specifying a path option.

Adding a library file

Steps 27 to 29 below are shown for reference only. No operation is required.

Step 27: Select [Linker] > [Libraries] from the [Build Options] tree.

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


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions
● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

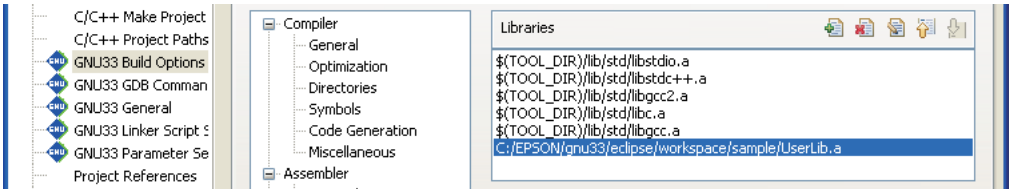
```
$(TOOL_DIR)/lib/std/libstdio.a
$(TOOL_DIR)/lib/std/libstdc++.a
$(TOOL_DIR)/lib/std/libgcc2.a
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a
```

The [Libraries] column lists the simulated I/O library, C++ library, `long long`-type emulation library, 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 28: 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.



```
$(TOOL_DIR)/lib/std/libstdio.a
$(TOOL_DIR)/lib/std/libstdc++.a
$(TOOL_DIR)/lib/std/libgcc2.a
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a
C:/EPSON/gnu33/eclipse/workspace/sample/UserLib.a
```

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

Step 29: Press the [Apply] button to confirm the changes made here.

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

Setting a linker script

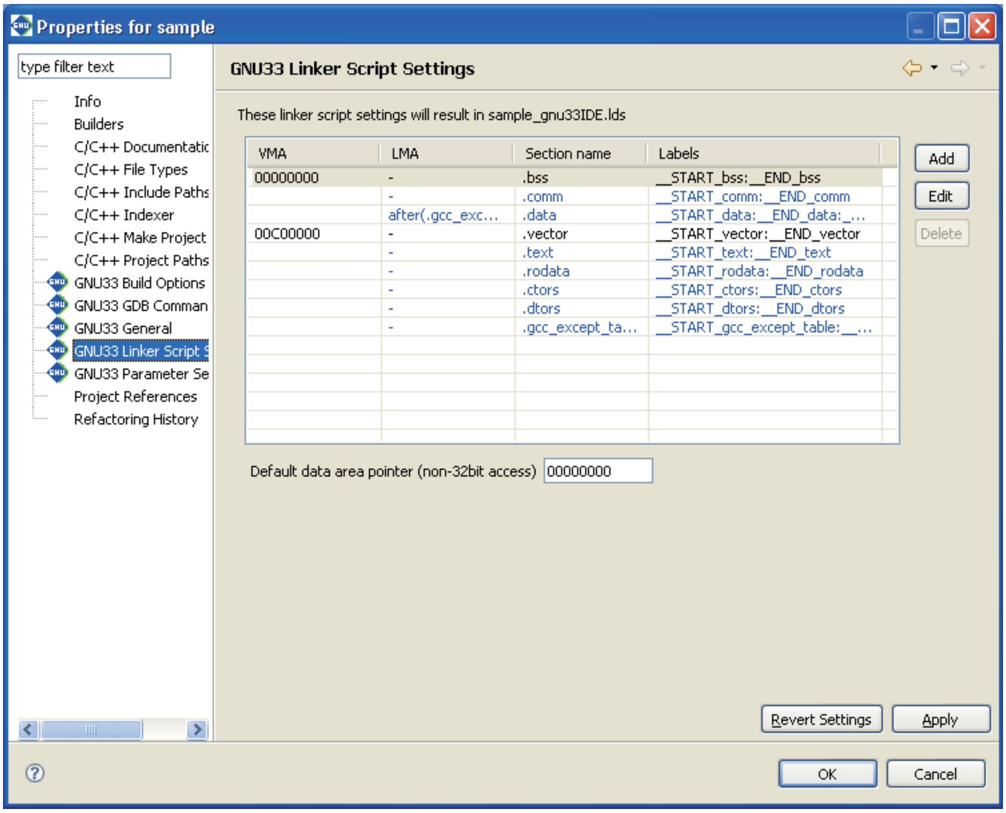
A build process requires a linker script file. This file also can be created with the **IDE**.

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 re-configuring 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 3.7, "Data Area and Sections".

Step 30: If you closed the [Properties] dialog box, select [Properties] from the [Project] menu to reopen it.

Step 31: Click to select [GNU33 Linker Script Settings] from the properties list.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Linker Script Settings
Add
Edit
Delete
Revert Settings
Apply
OK
Cancel

These linker script settings will result in sample_gnu33IDE.lds

VMA	LMA	Section name	Labels
00000000	-	.bss	__START_bss:__END_bss
	-	.comm	__START_comm:__END_comm
	after(.gcc_exc...	.data	__START_data:__END_data:_...
00C00000	-	.vector	__START_vector:__END_vector
	-	.text	__START_text:__END_text
	-	.rodata	__START_rodata:__END_rodata
	-	.ctors	__START_ctors:__END_ctors
	-	.dtors	__START_dtors:__END_dtors
	-	.gcc_except_ta...	__START_gcc_except_table:__...

Default data area pointer (non-32bit access) 00000000

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

`.bss`	Section in which variables without initial values are placed. (Normally located in RAM.)
`.comm`	Section in which variables without initial values are placed. (Normally located in RAM.)
`.data`	Section in which variables with initial values are placed. (The initial values are located in ROM. They are copied into RAM when needed.)
`.vector`	Section in which vector tables are placed. (The actual data is located in ROM.)
`.text`	Section in which program codes are placed. (The actual data is located in ROM and executed there or from high-speed RAM after copying.)
`.rodata`	Constant variables. (The actual data is located in ROM.)
`.ctors`	Pointer arrays to global class constructor functions. (The actual data is located in ROM.)
`.dtors`	Pointer arrays to global class destructor functions. (The actual data is located in ROM.)
`.gcc_except_table`	Table data for exception handling. (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(.gcc_except_table)`", it means that the actual data is to be located following the section indicated in parentheses (in this case, the `.gcc_except_table` 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.

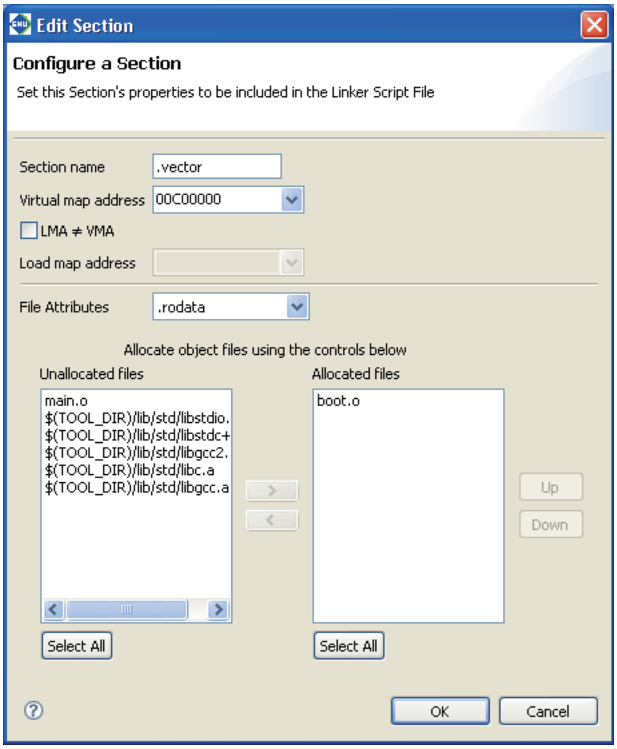
The section information is displayed in blue except for the `.vector` section displayed in black. Blue is used to display the standard sections defined by default and black is used to display other user defined sections. To edit the section name, standard section attribute, address to locate, and objects 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 that are located in the user sections with the same attribute.

The [Default data area pointer] text box is used to set the default data area pointer (address) that is to be written in a linker script file. The data area pointer is provided for the purpose of reducing the number of instructions by using one of the internal registers of the C33 Core as a pointer and accessing data within an address range confined by the base address stored in that register.
By default, the data pointer is set to 'base address 0' and will be output to the linker script file. For detailed information on data area, refer to Section 3.7, "Data Area and Sections".

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

The [Edit Section] dialog box will be displayed.



Edit Section

Configure a Section

Set this Section's properties to be included in the Linker Script File

Section name: .vector

Virtual map address: 00C00000

☐ LMA ≠ VMA

Load map address:

File Attributes: .rodata

Allocate object files using the controls below

Unallocated files:
- main.o
- $(TOOL_DIR)/lib/std/libstdio.
- $(TOOL_DIR)/lib/std/libstdc+
- $(TOOL_DIR)/lib/std/libgcc2.
- $(TOOL_DIR)/lib/std/libc.a
- $(TOOL_DIR)/lib/std/libgcc.a

Allocated files:
- boot.o

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/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 33: Click the [OK] button to close the dialog box.

Step 34: 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) as above 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

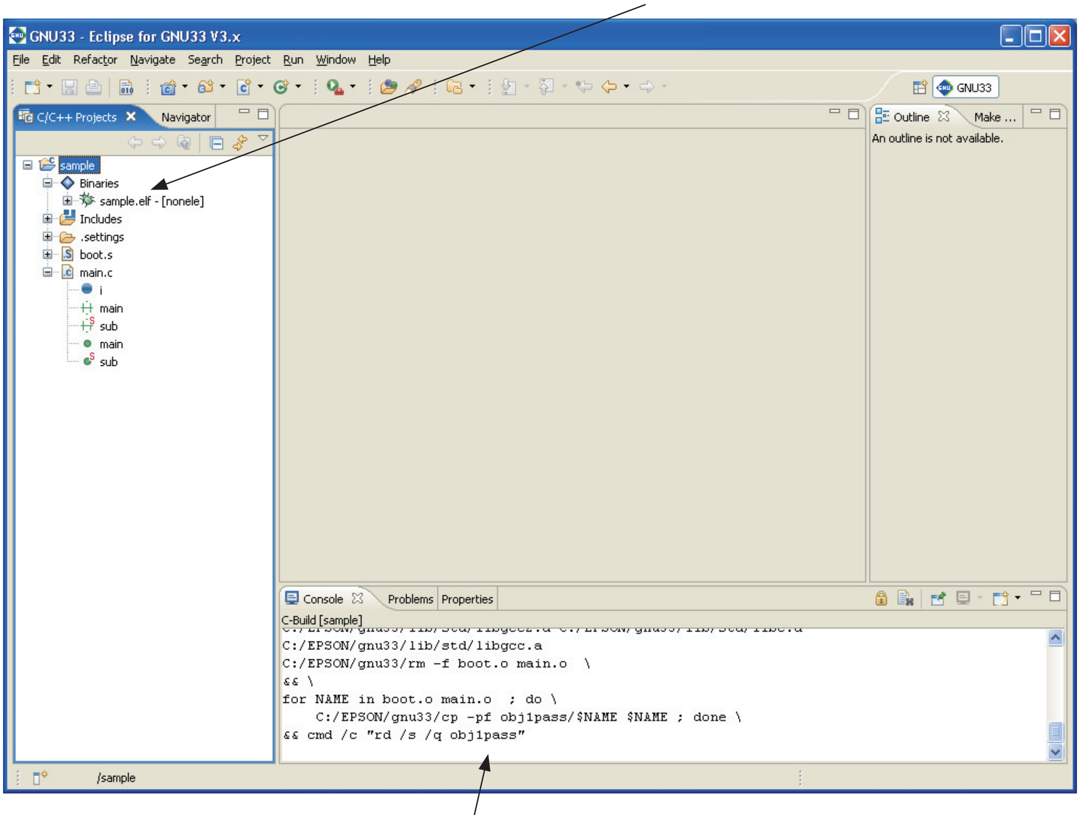
When you are finished with the work described in the preceding sections, you are ready to build (compile, assemble, and link) a program.

To execute a build process

Step 35: Select the project name "`sample`" from the [C/C++ Projects] view.

Step 36: 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`.


GNU33 - Eclipse for GNU33 V3.x
File Edit Refactor Navigate Search Project Run Window Help
GNU33
C/C++ Projects
Navigator
sample
Binaries
sample.elf - [nonele]
Includes
.settings
boot.s
main.c
i
main
sub
main
sub
Outline
Make ...
An outline is not available.
Console
Problems
Properties
/sample

C-Build [sample]

```
C:/EPSON/gnu33/lib/std/libgcc.a
C:/EPSON/gnu33/rm -f boot.o main.o  \
&& \
for NAME in boot.o main.o  ; do \
    C:/EPSON/gnu33/cp -pf obj1pass/$NAME $NAME ; done \
&& cmd /c "rd /s /q obj1pass"
```

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

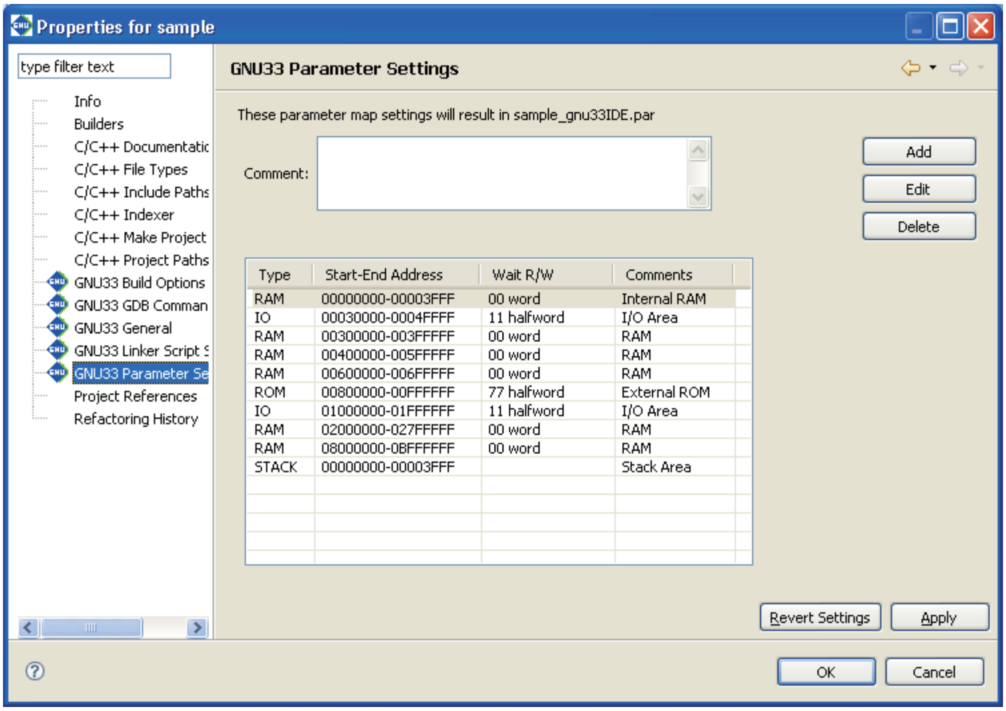
3.3.6 Debugging a Program

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

Step 38: Select [GNU33 Parameter Settings] from the properties list by clicking on it.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
Add
Edit
Delete
Revert Settings
Apply
OK
Cancel

GNU33 Parameter Settings

These parameter map settings will result in sample_gnu33IDE.par

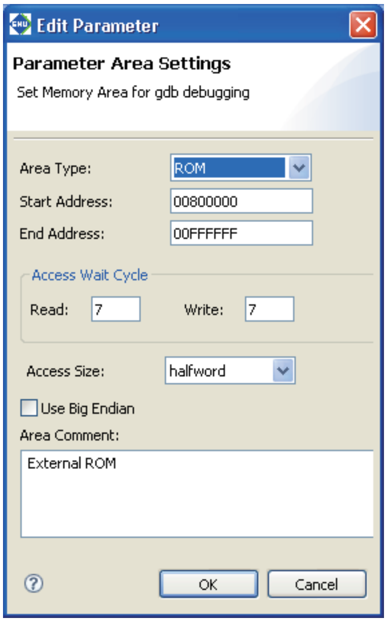
Comment:

Type	Start-End Address	Wait R/W	Comments
RAM	00000000-00003FFF	00 word	Internal RAM
IO	00030000-0004FFFF	11 halfword	I/O Area
RAM	00300000-003FFFFF	00 word	RAM
RAM	00400000-005FFFFF	00 word	RAM
RAM	00600000-006FFFFF	00 word	RAM
ROM	00800000-00FFFFFF	77 halfword	External ROM
IO	01000000-01FFFFFF	11 halfword	I/O Area
RAM	02000000-027FFFFF	00 word	RAM
RAM	08000000-0BFFFFFF	00 word	RAM
STACK	00000000-00003FFF		Stack Area

Ten items of area information that have been set by default will be displayed.
The information for RAM at the top, for example, defines that 0x0 to 0x3fff (16K bytes) in area 0 be used as a RAM area. Note that "`00 word`" here means this device is accessed wordwise 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.)
Similarly, other area information for the I/O memory, RAM, and ROM areas and finally the stack area in RAM are defined.
Shown here is the basic configuration of the S1C33 microcomputer that incorporates the C33 STD or C33 PE 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 more than the default, a parameter file may be created directly as shown here without incurring any problem.
As for an example, we'll change the start address of the ROM area (0x800000–0xffffff) to 0xc00000 and change the number of wait states to 4.

Step 39: Click on the ROM 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.


Edit Parameter
OK
Cancel

Parameter Area Settings

Set Memory Area for gdb debugging

Area Type: ROM
Start Address: 00800000
End Address: 00FFFFFF

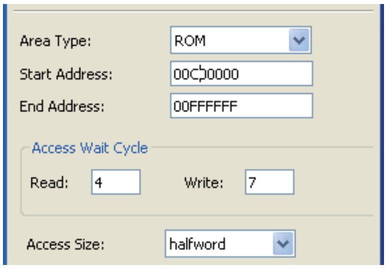
Access Wait Cycle
Read: 7 Write: 7

Access Size: halfword

☐ Use Big Endian

Area Comment: External ROM

Step 40: Enter 00C00000 in the [Start Address:] text box and 4 (4 wait states) in the [Read:] text box.

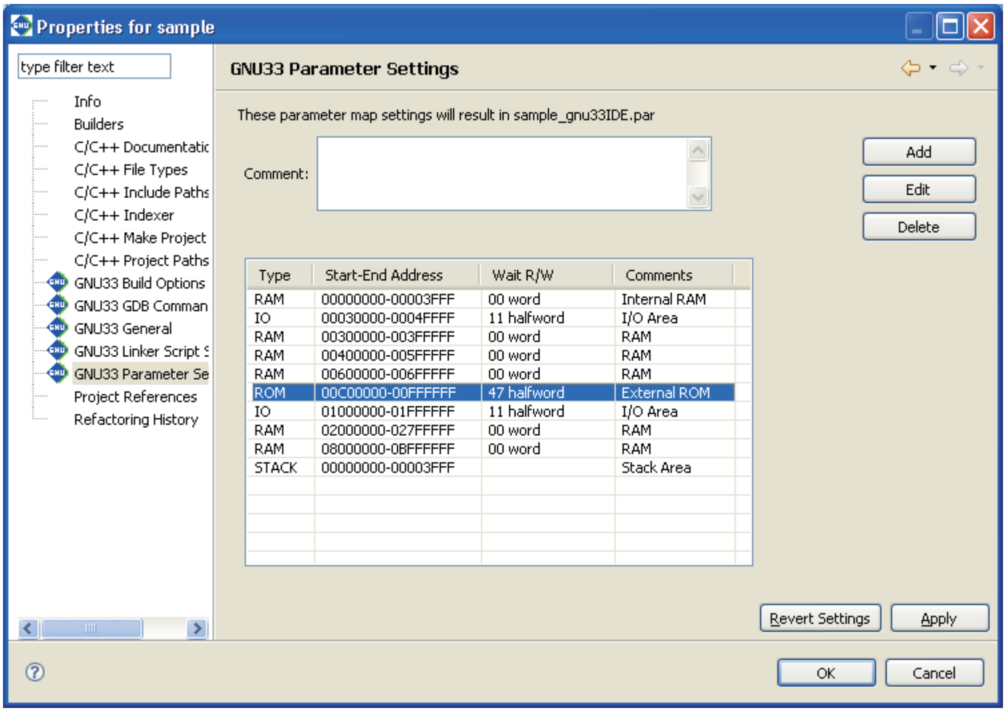


Area Type: ROM
Start Address: 00C00000
End Address: 00FFFFFF

Access Wait Cycle
Read: 4 Write: 7

Access Size: halfword

Step 41: Click the [OK] button.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
Add
Edit
Delete
Revert Settings
Apply
OK
Cancel

GNU33 Parameter Settings

These parameter map settings will result in sample_gnu33IDE.par

Comment:

Type	Start-End Address	Wait R/W	Comments
RAM	00000000-00003FFF	00 word	Internal RAM
IO	00030000-0004FFFF	11 halfword	I/O Area
RAM	00300000-003FFFFF	00 word	RAM
RAM	00400000-005FFFFF	00 word	RAM
RAM	00600000-006FFFFF	00 word	RAM
ROM	00C00000-00FFFFFF	47 halfword	External ROM
IO	01000000-01FFFFFF	11 halfword	I/O Area
RAM	02000000-027FFFFF	00 word	RAM
RAM	08000000-0BFFFFFF	00 word	RAM
STACK	00000000-00003FFF		Stack Area

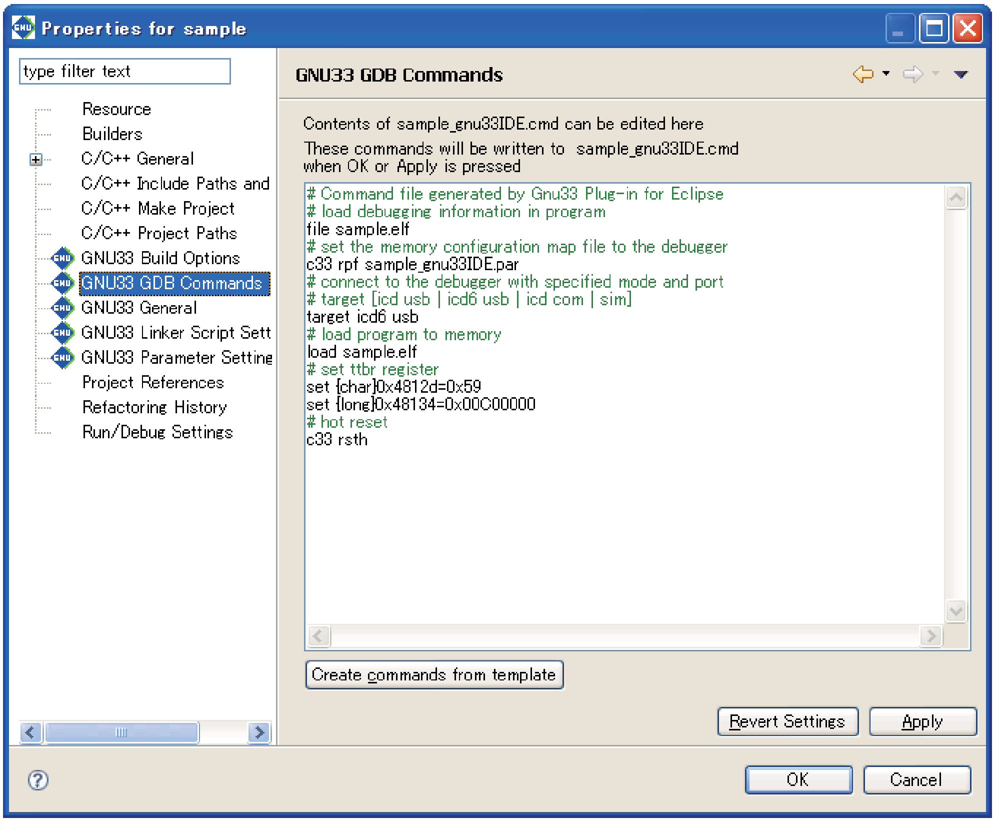
The displayed address range and the access conditions of the area have been changed to "`00C00000-00FFFFFF`" and to "`47 halfword`", respectively.

Step 42: Click the [Apply] button.

When above settings are made, a file named "`sample_gnu33IDE.par`" is generated and passed to the debugger via a command file when the debugger starts.

Setting the debugger's startup options

Step 43: Select [GNU33 GDB Commands] from the properties list by clicking on it.


Properties for sample
type filter text
Resource
Builders
C/C++ General
C/C++ Include Paths and
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Commands
GNU33 General
GNU33 Linker Script Sett
GNU33 Parameter Setting
Project References
Refactoring History
Run/Debug Settings
GNU33 GDB Commands
Create commands from template
Revert Settings
Apply
OK
Cancel

Contents of sample_gnu33IDE.cmd can be edited here

These commands will be written to sample_gnu33IDE.cmd when OK or Apply is pressed

```
# Command file generated by Gnu33 Plug-in for Eclipse
# load debugging information in program
file sample.elf
# set the memory configuration map file to the debugger
c33 rpf sample_gnu33IDE.par
# connect to the debugger with specified mode and port
# target [icd usb | icd6 usb | icd com | sim]
target icd6 usb
# load program to memory
load sample.elf
# set ttbr register
set {char}0x4812d=0x59
set {long}0x48134=0x00C00000
# hot reset
c33 rsth
```

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

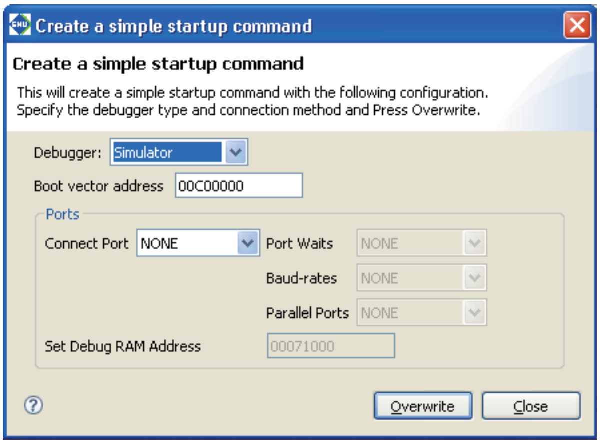
The debugger must be set to the appropriate mode that suits the ICD used, etc. before it can be operated. For detailed information, refer to Section 3.6, "Debugging Environment". Here, we'll set the debugger to simulator mode that does not require external equipment before we start debugging.

The contents of the command file displayed by default are provided for debugging using an ICD Ver.6.

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

Step 44: Click the [Create commands from template] button to display the [Create a simple startup command] dialog box.

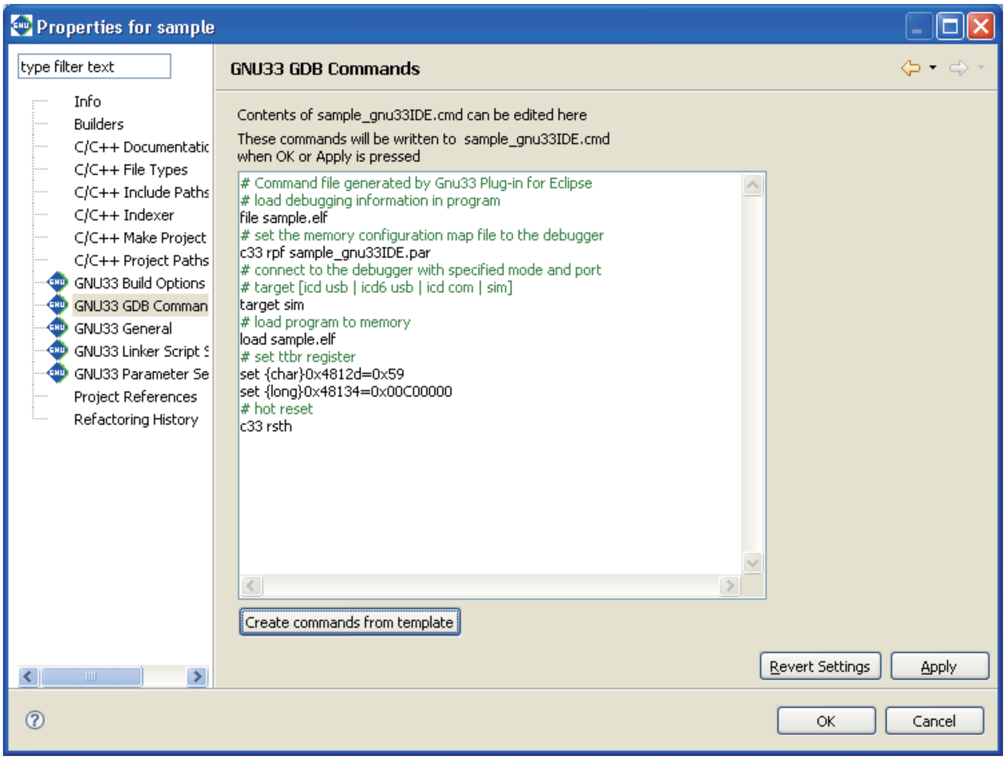
Step 45: Select "Simulator" from the [Debugger:] combo box.


Create a simple startup command
Create a simple startup command
Ports
Overwrite
Close

This will create a simple startup command with the following configuration.
Specify the debugger type and connection method and Press Overwrite.

Debugger: Simulator
Boot vector address 00C00000
Connect Port NONE
Port Waits NONE
Baud-rates NONE
Parallel Ports NONE
Set Debug RAM Address 00071000

Step 46: Click the [Overwrite] button.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 GDB Commands
Create commands from template
Revert Settings
Apply
OK
Cancel

Contents of sample_gnu33IDE.cmd can be edited here

These commands will be written to sample_gnu33IDE.cmd when OK or Apply is pressed

```
# Command file generated by Gnu33 Plug-in for Eclipse
# load debugging information in program
file sample.elf
# set the memory configuration map file to the debugger
c33 rpf sample_gnu33IDE.par
# connect to the debugger with specified mode and port
# target [icd usb | icd6 usb | icd com | sim]
target sim
# load program to memory
load sample.elf
# set ttbr register
set {char}0x4812d=0x59
set {long}0x48134=0x00C00000
# hot reset
c33 rsth
```

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

Step 47: Click the [OK] button.

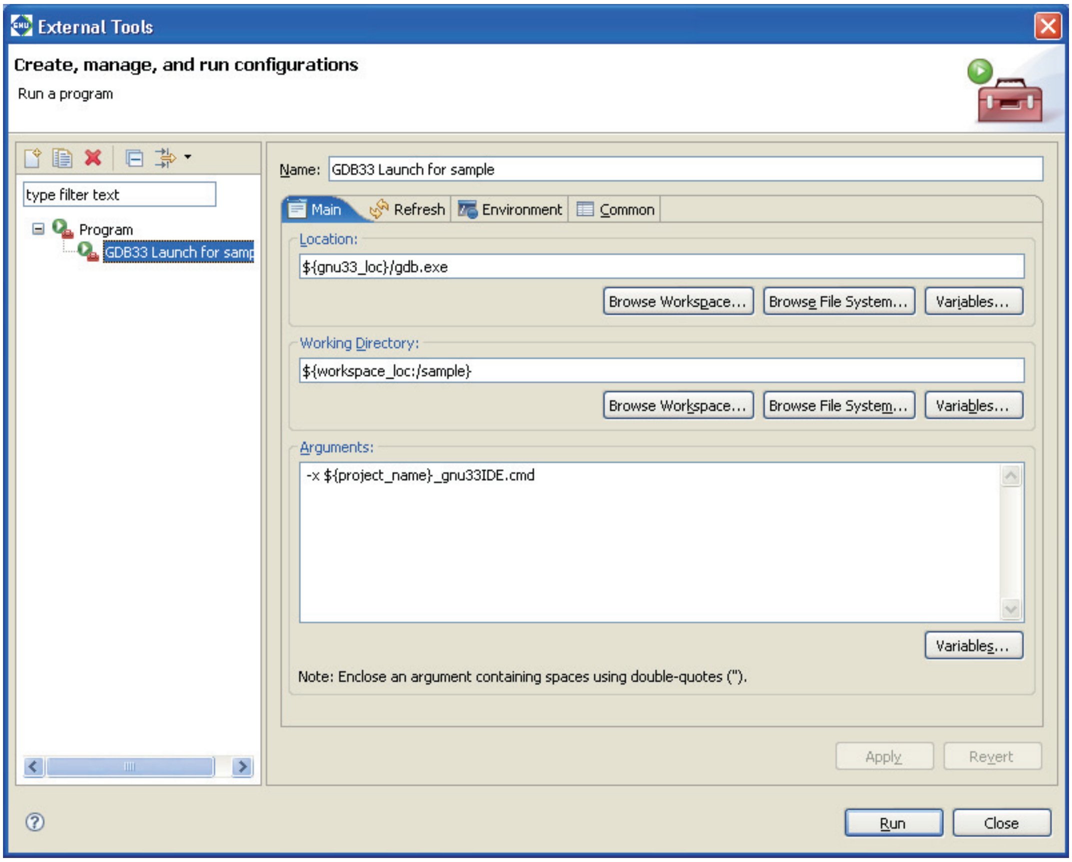
When above settings are made, a command file named "`sample_gnu33IDE.cmd`" is generated and it will be passed to the debugger.

Starting the debugger

Step 48: Select [External Tools] > [External Tools Configurations...] from the [Run] menu.

The [External Tools Configurations] dialog box will be displayed.

Step 49: Select [GDB33 Launch for sample] from the list.



External Tools
Create, manage, and run configurations
Run a program
type filter text
Program
GDB33 Launch for samp
Main
Refresh
Environment
Common
Browse Workspace...
Browse File System...
Variables...
Apply
Revert
Run
Close

Name: GDB33 Launch for sample

Location: ${gnu33_loc}/gdb.exe

Working Directory: ${workspace_loc:/sample}

Arguments: -x ${project_name}_gnu33IDE.cmd

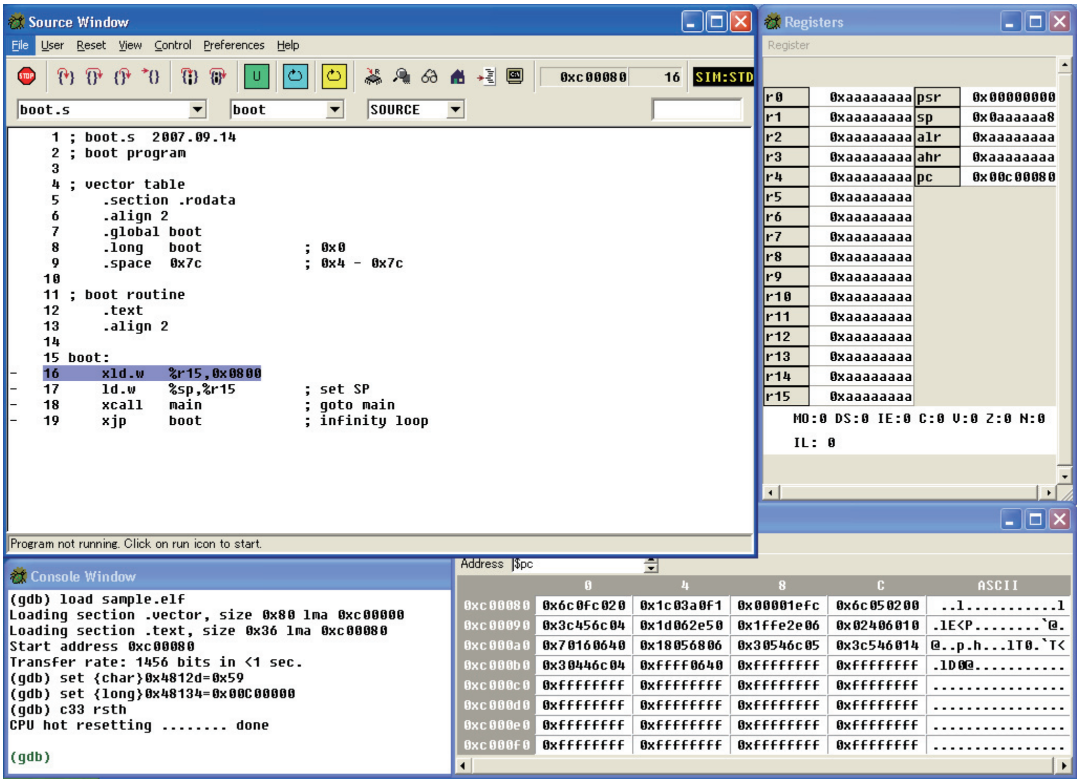
Note: Enclose an argument containing spaces using double-quotes (").

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 50: 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 [GNU33 GDB Commands] dialog box.



```
; boot.s  2007.09.14
; boot program

; vector table
    .section .rodata
    .align 2
    .global boot
    .long   boot                ; 0x0
    .space  0x7c                ; 0x4 - 0x7c

; boot routine
    .text
    .align 2

boot:
    xld.w   %r15,0x0800
    ld.w    %sp,%r15            ; set SP
    xcall   main                ; goto main
    xjp     boot                ; infinity loop
```

Program not running. Click on run icon to start.

```
r0      0xaaaaaaaa psr   0x00000000
r1      0xaaaaaaaa sp    0x0aaaaaa8
r2      0xaaaaaaaa alr   0xaaaaaaaa
r3      0xaaaaaaaa ahr   0xaaaaaaaa
r4      0xaaaaaaaa pc    0x00c00080
r5      0xaaaaaaaa
r6      0xaaaaaaaa
r7      0xaaaaaaaa
r8      0xaaaaaaaa
r9      0xaaaaaaaa
r10     0xaaaaaaaa
r11     0xaaaaaaaa
r12     0xaaaaaaaa
r13     0xaaaaaaaa
r14     0xaaaaaaaa
r15     0xaaaaaaaa
   MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
   IL: 0
```

```
(gdb) load sample.elf
Loading section .vector, size 0x80 lma 0xc00000
Loading section .text, size 0x36 lma 0xc00080
Start address 0xc00080
Transfer rate: 1456 bits in <1 sec.
(gdb) set {char}0x4812d=0x59
(gdb) set {long}0x48134=0x00C00000
(gdb) c33 rsth
CPU hot resetting ........ done

(gdb)
```

Address $pc

```
                  0          4          8          C          ASCII
0xc00080   0x6c0fc020 0x1c03a0f1 0x00001efc 0x6c050200  ..l...........l
0xc00090   0x3c456c04 0x1d062e50 0x1ffe2e06 0x02406010 .lE<P........`@.
0xc000a0   0x70160640 0x18056806 0x30546c05 0x3c546014 @..p.h...lT0.`T<
0xc000b0   0x30446c04 0xffff0640 0xffffffff 0xffffffff .lD0@...........
0xc000c0   0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
0xc000d0   0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
0xc000e0   0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
0xc000f0   0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
```

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

To run a program

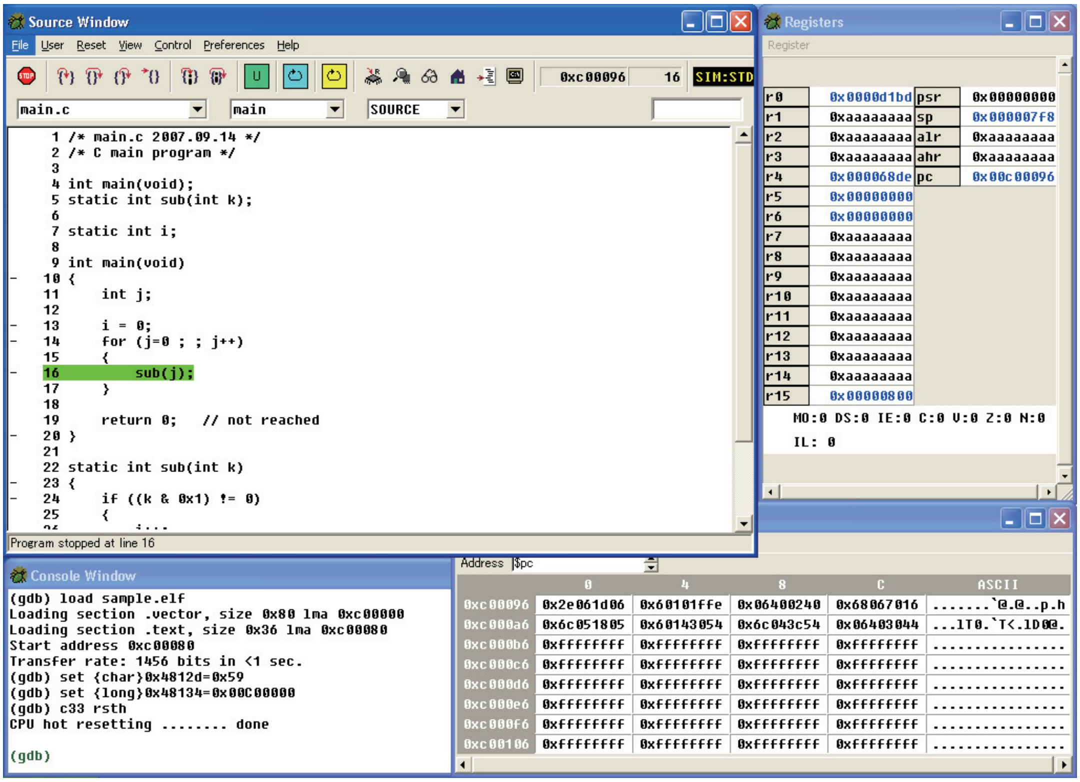
Step 51: Click the [Continue] button in the toolbar.

The sample program here endlessly increments the int variable counter '`i`'.

Use a forcible break to stop such an endless loop.

To forcibly break a program

Step 52: Click the [Stop] button in the toolbar.



```
    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     {
```

Program stopped at line 16

```
r0   0x0000d1bd  psr  0x00000000
r1   0xaaaaaaaa  sp   0x000007f8
r2   0xaaaaaaaa  alr  0xaaaaaaaa
r3   0xaaaaaaaa  ahr  0xaaaaaaaa
r4   0x000068de  pc   0x00c00096
r5   0x00000000
r6   0x00000000
r7   0xaaaaaaaa
r8   0xaaaaaaaa
r9   0xaaaaaaaa
r10  0xaaaaaaaa
r11  0xaaaaaaaa
r12  0xaaaaaaaa
r13  0xaaaaaaaa
r14  0xaaaaaaaa
r15  0x00000800
  MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
  IL: 0
```

```
(gdb) load sample.elf
Loading section .vector, size 0x80 lma 0xc00000
Loading section .text, size 0x36 lma 0xc00080
Start address 0xc00080
Transfer rate: 1456 bits in <1 sec.
(gdb) set {char}0x4812d=0x59
(gdb) set {long}0x48134=0x00C00000
(gdb) c33 rsth
CPU hot resetting ........ done

(gdb)
```

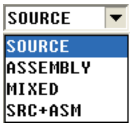
```
          0          4          8          C          ASCII
0xc00096  0x2e061d06 0x60101ffe 0x06400240 0x68067016 .......`@.@..p.h
0xc000a6  0x6c051805 0x60143054 0x6c043c54 0x06403044 ...1T0.`T<.1D0@.
0xc000b6  0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
0xc000c6  0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
0xc000d6  0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
0xc000e6  0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
0xc000f6  0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
0xc00106  0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
```

Notice that the "`sub(j);`" 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 0x00c00096. 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

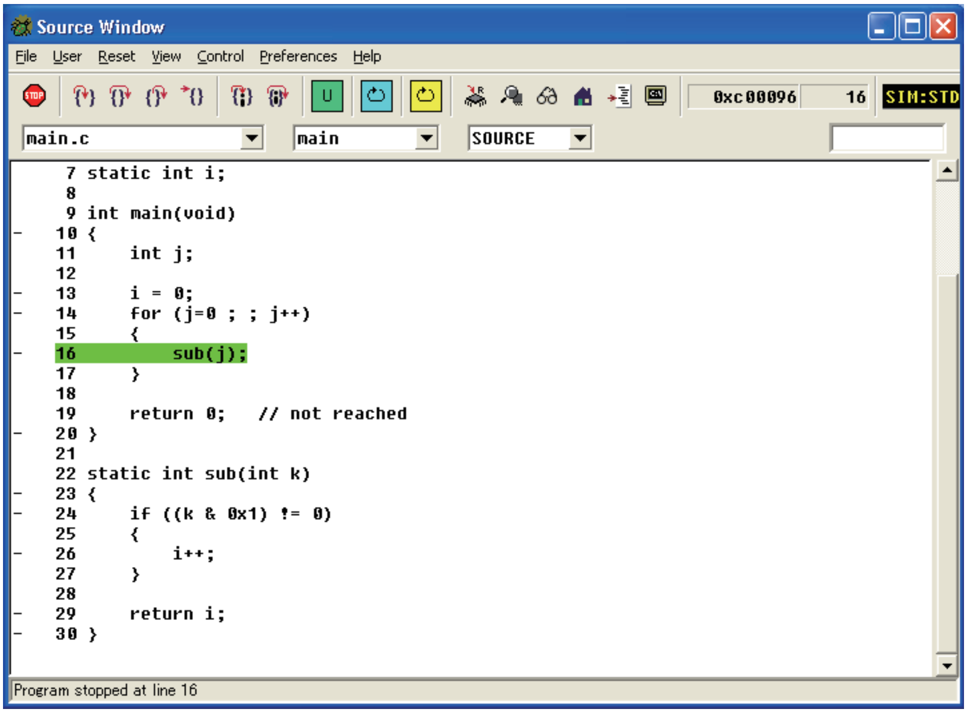
There is the pulldown list box that shows SOURCE in the toolbar. Use it to specify the display mode of the [Source] window.



Step 53: 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.

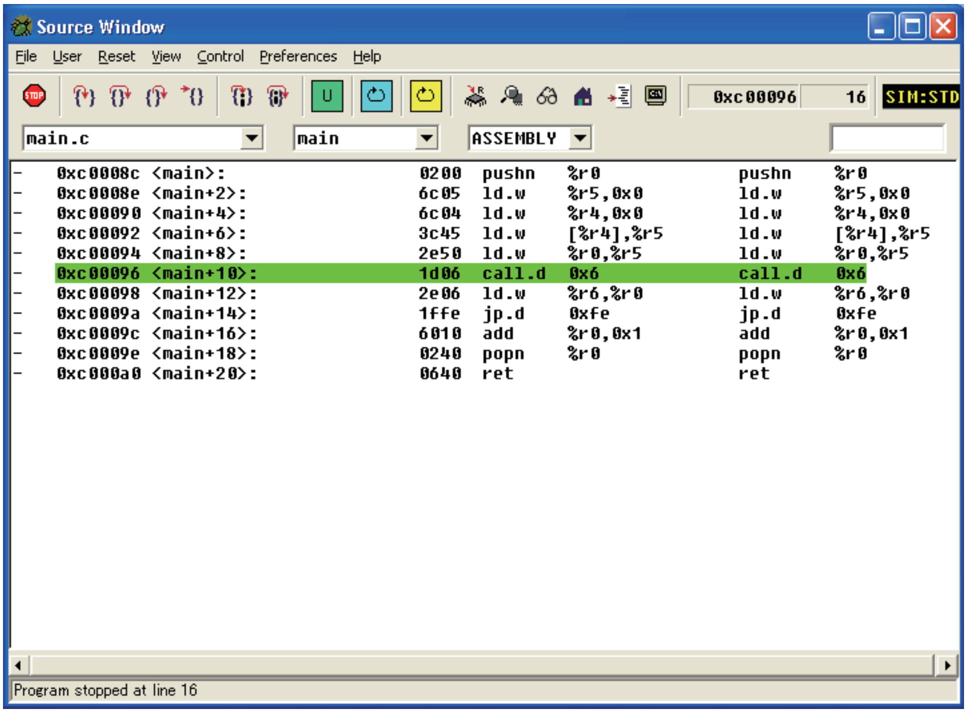

Source Window
File
User
Reset
View
Control
Preferences
Help
0xc00096
16
SIM:STD
main.c
main
SOURCE

```
      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     }
     28
-    29     return i;
-    30 }
```

Program stopped at line 16

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.

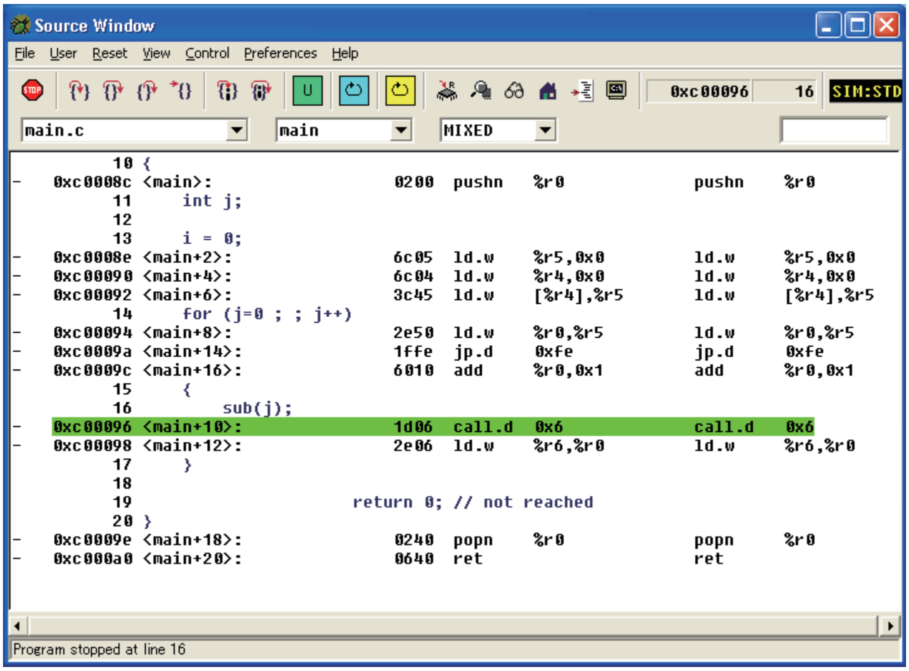

Source Window
File
User
Reset
View
Control
Preferences
Help
0xc00096
16
SIM:STD
main.c
main
ASSEMBLY

```
-    0xc0008c <main>:                0200  pushn   %r0           pushn   %r0
-    0xc0008e <main+2>:              6c05  ld.w    %r5,0x0       ld.w    %r5,0x0
-    0xc00090 <main+4>:              6c04  ld.w    %r4,0x0       ld.w    %r4,0x0
-    0xc00092 <main+6>:              3c45  ld.w    [%r4],%r5     ld.w    [%r4],%r5
-    0xc00094 <main+8>:              2e50  ld.w    %r0,%r5       ld.w    %r0,%r5
-    0xc00096 <main+10>:             1d06  call.d  0x6           call.d  0x6
-    0xc00098 <main+12>:             2e06  ld.w    %r6,%r0       ld.w    %r6,%r0
-    0xc0009a <main+14>:             1ffe  jp.d    0xfe          jp.d    0xfe
-    0xc0009c <main+16>:             6010  add     %r0,0x1       add     %r0,0x1
-    0xc0009e <main+18>:             0240  popn    %r0           popn    %r0
-    0xc000a0 <main+20>:             0640  ret                   ret
```

Program stopped at line 16

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.

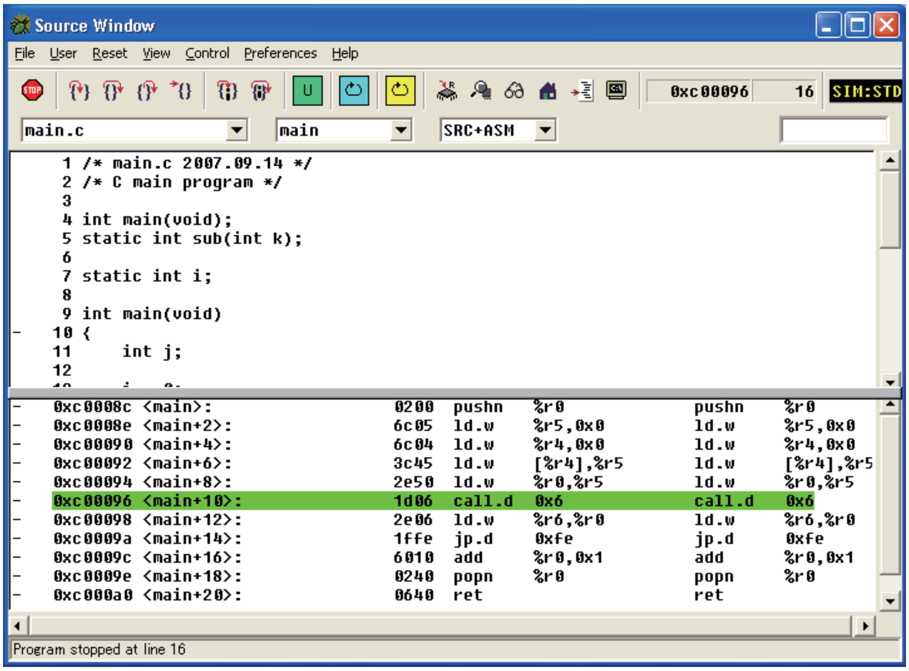

Source Window
File User Reset View Control Preferences Help
STOP
U
0xc00096
16
SIM:STD
main.c
main
MIXED

```
        10 {
- 0xc0008c <main>:                 0200  pushn   %r0           pushn    %r0
        11     int j;
        12
        13     i = 0;
- 0xc0008e <main+2>:               6c05  ld.w    %r5,0x0       ld.w     %r5,0x0
- 0xc00090 <main+4>:               6c04  ld.w    %r4,0x0       ld.w     %r4,0x0
- 0xc00092 <main+6>:               3c45  ld.w    [%r4],%r5     ld.w     [%r4],%r5
        14     for (j=0 ; ; j++)
- 0xc00094 <main+8>:               2e50  ld.w    %r0,%r5       ld.w     %r0,%r5
- 0xc0009a <main+14>:              1ffe  jp.d    0xfe          jp.d     0xfe
- 0xc0009c <main+16>:              6010  add     %r0,0x1       add      %r0,0x1
        15     {
        16         sub(j);
- 0xc00096 <main+10>:              1d06  call.d  0x6           call.d   0x6
- 0xc00098 <main+12>:              2e06  ld.w    %r6,%r0       ld.w     %r6,%r0
        17     }
        18
        19                     return 0; // not reached
        20 }
- 0xc0009e <main+18>:              0240  popn    %r0           popn     %r0
- 0xc000a0 <main+20>:              0640  ret                   ret
```

Program stopped at line 16

SRC+ASM

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


Source Window
File User Reset View Control Preferences Help
STOP
U
0xc00096
16
SIM:STD
main.c
main
SRC+ASM

```
       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
```

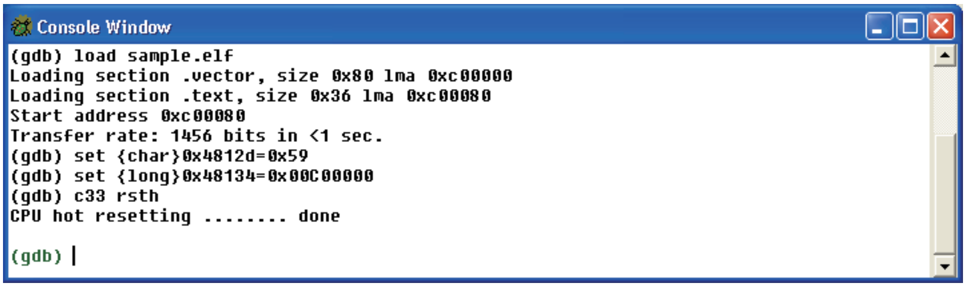
```
- 0xc0008c <main>:                 0200  pushn   %r0           pushn    %r0
- 0xc0008e <main+2>:               6c05  ld.w    %r5,0x0       ld.w     %r5,0x0
- 0xc00090 <main+4>:               6c04  ld.w    %r4,0x0       ld.w     %r4,0x0
- 0xc00092 <main+6>:               3c45  ld.w    [%r4],%r5     ld.w     [%r4],%r5
- 0xc00094 <main+8>:               2e50  ld.w    %r0,%r5       ld.w     %r0,%r5
- 0xc00096 <main+10>:              1d06  call.d  0x6           call.d   0x6
- 0xc00098 <main+12>:              2e06  ld.w    %r6,%r0       ld.w     %r6,%r0
- 0xc0009a <main+14>:              1ffe  jp.d    0xfe          jp.d     0xfe
- 0xc0009c <main+16>:              6010  add     %r0,0x1       add      %r0,0x1
- 0xc0009e <main+18>:              0240  popn    %r0           popn     %r0
- 0xc000a0 <main+20>:              0640  ret                   ret
```

Program stopped at line 16

Step 54: Select SOURCE to reverse the display mode.

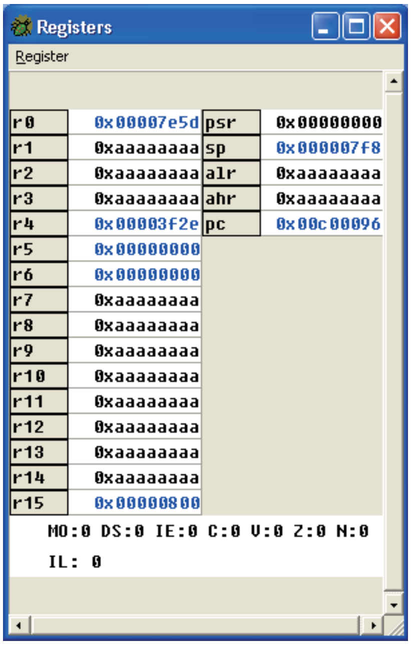
Let's take a look at other windows of the debugger here.

About the debugger windows



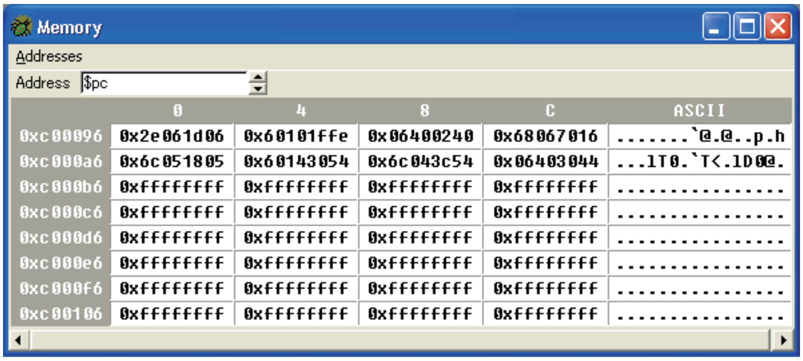
```
(gdb) load sample.elf
Loading section .vector, size 0x80 lma 0xc00000
Loading section .text, size 0x36 lma 0xc00080
Start address 0xc00080
Transfer rate: 1456 bits in <1 sec.
(gdb) set {char}0x4812d=0x59
(gdb) set {long}0x48134=0x00C00000
(gdb) c33 rsth
CPU hot resetting ........ done

(gdb) |
```

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 51, the same effect can be achieved by entering `cont` here and pressing the [Enter] key.



```
r0   0x00007e5d psr 0x00000000
r1   0xaaaaaaaa sp  0x000007f8
r2   0xaaaaaaaa alr 0xaaaaaaaa
r3   0xaaaaaaaa ahr 0xaaaaaaaa
r4   0x00003f2e pc  0x00c00096
r5   0x00000000
r6   0x00000000
r7   0xaaaaaaaa
r8   0xaaaaaaaa
r9   0xaaaaaaaa
r10  0xaaaaaaaa
r11  0xaaaaaaaa
r12  0xaaaaaaaa
r13  0xaaaaaaaa
r14  0xaaaaaaaa
r15  0x00000800
   MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
   IL: 0
```

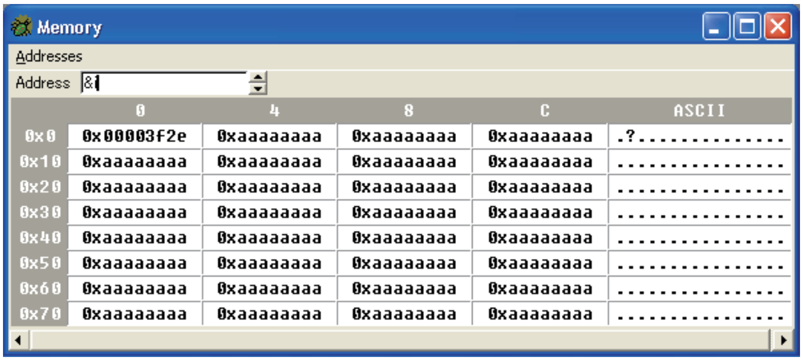
This is the [Registers] window. It shows the contents of the CPU registers. The register data can be rewritten here.



```
Address $pc
           0          4          8          C          ASCII
0xc00096   0x2e061d06 0x60101ffe 0x06400240 0x68067016 .......`@.@..p.h
0xc000a6   0x6c051805 0x60143054 0x6c043c54 0x06403044 ...lT0.`T<.lD0@.
0xc000b6   0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
0xc000c6   0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
0xc000d6   0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
0xc000e6   0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
0xc000f6   0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
0xc00106   0xffffffff 0xffffffff 0xffffffff 0xffffffff ................
```

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 0x0000000–0x0000003). Examine value of the variable '`i`' in the [Memory] window.

Step 55: Enter '`&i`' or '`0`' in the [Address] text box of the [Memory] window and press the [Enter] key.



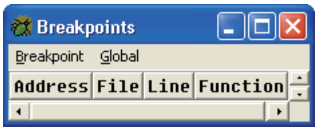
Address &i

```
       0           4           8           C           ASCII
0x0    0x00003f2e  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  .?..............
0x10   0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
0x20   0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
0x30   0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
0x40   0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
0x50   0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
0x60   0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
0x70   0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
```

The memory contents are displayed, beginning with the variable '`i`' (address 0). As shown here, '`i`' has been counted up to 0x00003f2e (= 16,174).

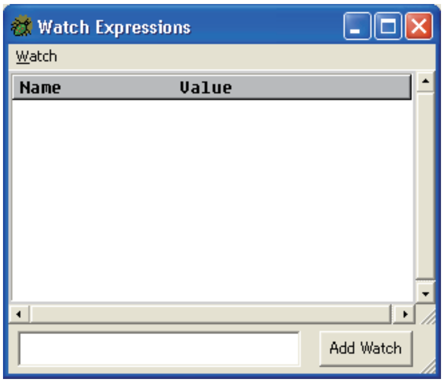
These three windows are displayed by default when the debugger starts, along with the [Source] window.
In addition, the following three windows are available.

Step 56: Click the [Breakpoints] button.



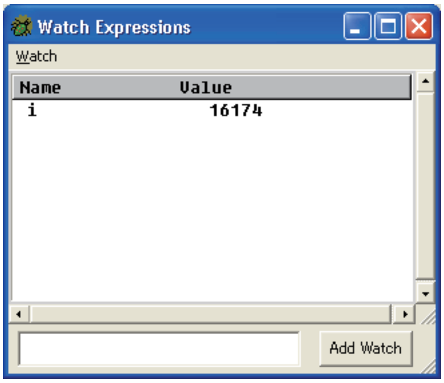
This is the [Breakpoints] window. This window is used to manage software PC breakpoints that halt the program at specified positions.

Step 57: 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.

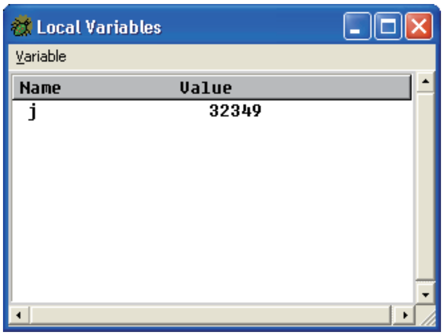
Step 58: Enter '`i`' in the text box located at the bottom of the window and click the [Add Watch] button.



Name	Value
i	16174

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 59: Click the [Local Variables] button.



Name	Value
j	32349

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.

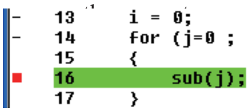
We have thus far seen the windows for the **gdb** debugger. Each window has other facilities, not just the ones that display information. These are detailed in Section 10.4, "Windows".

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 60: 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 13, click there to



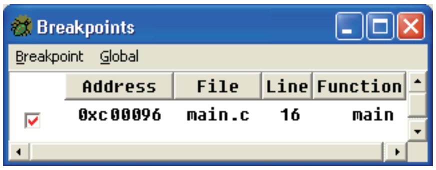
```
-    13      i = 0;
-    14      for (j=0 ; ; j++)
     15      {
■    16          sub(j);
     17      }
```

Step 61: Click the [Continue] button.

In contrast to Step 51, 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 62: Click the [Breakpoints] button.



	Address	File	Line	Function
☑	0xc00096	main.c	16	main

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 63: 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, and a data break that causes the program to stop upon accessing a specified address. 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 64: 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 64, 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 65: Click the [Next] button in the toolbar.

Repeat Step 65 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.

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 66: 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 other modes after connecting the in-circuit debugger S5U1C33000H or S5U1C33001H to the target board or by using the target board that incorporates the debug monitor (S5U1C30M2S) in it with the S5U1C330M1D1. For detailed information on how to debug in these modes, refer to Section 3.6, "Debugging Environment".

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

To quit the IDE

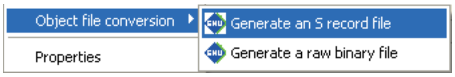
Step 67: Select [Exit] from the **IDE**'s [File] menu.

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 68: 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\gnu33>objcopy -I elf32-little -O srec --srec-forceS3 InputFile (.elf) OutputFile (.sa)
```

`-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\gnu33>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 mask data by **moto2ff.exe**.
Open the command prompt window and execute **moto2ff** in the command line as shown below.

```
C:\EPSON\gnu33>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 0x100000 bytes of data from the address 0x600000 in Motorola S3 format `input.sa` to a file `input.saf`

```
C:\EPSON\gnu33>moto2ff 600000 100000 input.sa
```

All data in the address range 0x600000 to 0x6fffff in `input.sa` will be output to a file. Any blank address in this address range is filled with 0xff data.

After creating a mask ROM file for the internal ROM according to the above procedure, be sure to perform the final verification of program operation using that file.
After verification, rename the mask ROM file to the one specified by Seiko Epson before presenting it to Seiko Epson.

3.4 Tutorial 2 (Using the User Makefiles)

In this section, as an example for using user makefiles, we'll take a look at a series of procedures, from building a project in the **IDE** to starting the debugger using the Toppers33 sample program and makefiles created for use with the old gnu33 tools. For basic information on using the **IDE**, etc., refer to Tutorial 1.

Sample directories used

This discussion assumes that Toppers33 and gnu33 tools are installed in the directories shown below.
Toppers33 installation directory: `C:\EPSON\gnu33\tool\tps33g`
gnu33 installation directory: `C:\EPSON\gnu33`

Note: This sample program is created as suitable for debug environment ICD Ver. 3 and later. In this tutorial, the explanation is made assuming that the program is run in ICD Ver. 3 and DMT33209 debug environment. However, since this tutorial aims to explain operations for up to starting the debugger, the environment is not required unless you actually want to execute the program.
For other details such as the supported type of demonstration board, connections with the ICD, and board settings, refer to `readme_s1c33.txt` (in English) or `readme_ja_s1c33.txt` (in Japanese) in the `C:\EPSON\gnu33\tool\tps33g` directory.

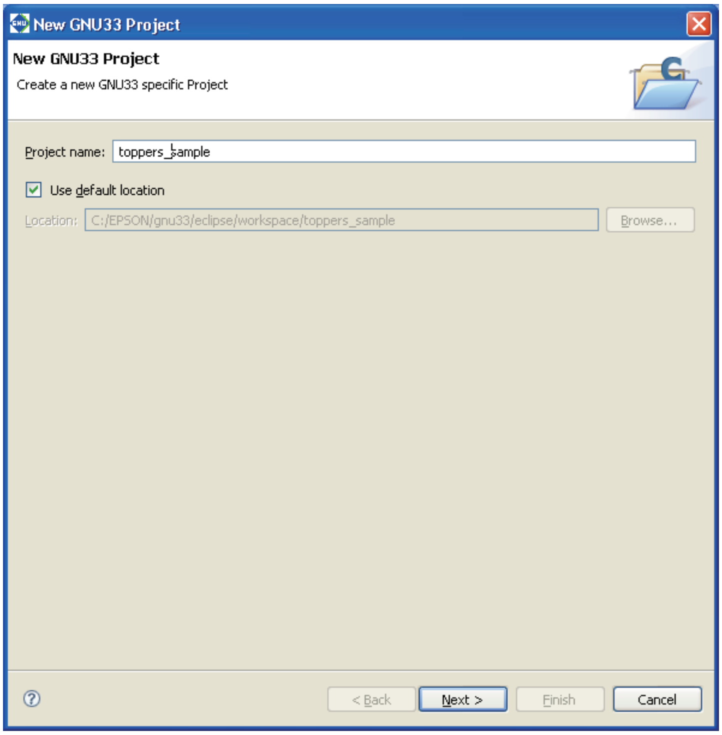
3.4.1 Creating a Project

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

Step 1: Launch the **IDE**.

Step 2: Select [New] > [New GNU33 Project] from the [File] menu to start the [New GNU33 Project] wizard.

Step 3: Enter the project name "`toppers_sample`" in the [Project name:] field.



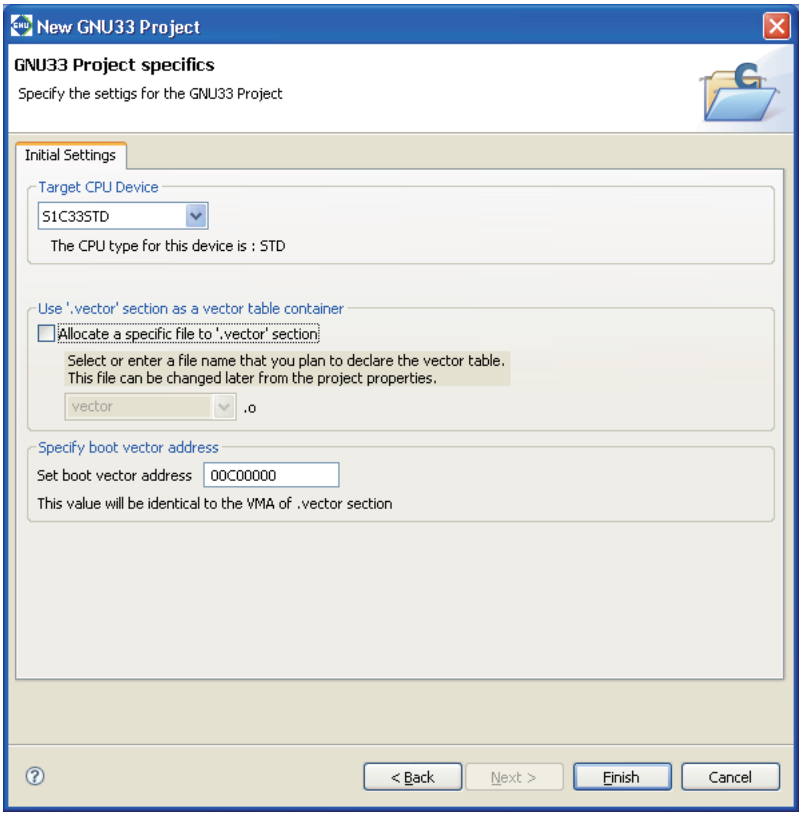
Project name: toppers_sample

☑ Use default location

Location: C:/EPSON/gnu33/eclipse/workspace/toppers_sample

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.



New GNU33 Project

GNU33 Project specifics

Specify the settigs for the GNU33 Project

Initial Settings

Target CPU Device

S1C33STD

The CPU type for this device is : STD

Use '.vector' section as a vector table container

☐ Allocate a specific file to '.vector' section

Select or enter a file name that you plan to declare the vector table.
This file can be changed later from the project properties.

vector .o

Specify boot vector address

Set boot vector address 00C00000

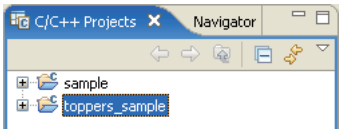
This value will be identical to the VMA of .vector section

< Back
Next >
Finish
Cancel

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 and the `.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.



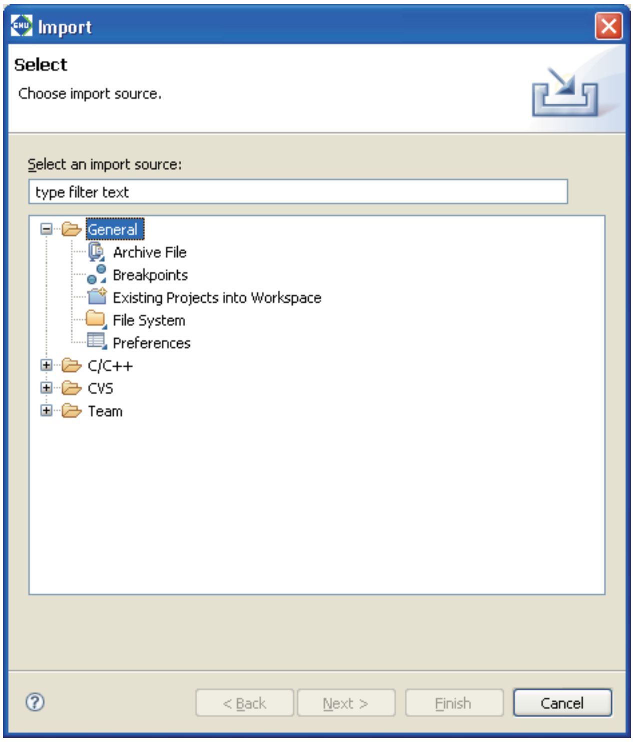
C/C++ Projects
Navigator
sample
toppers_sample

3.4.2 Importing Source Files

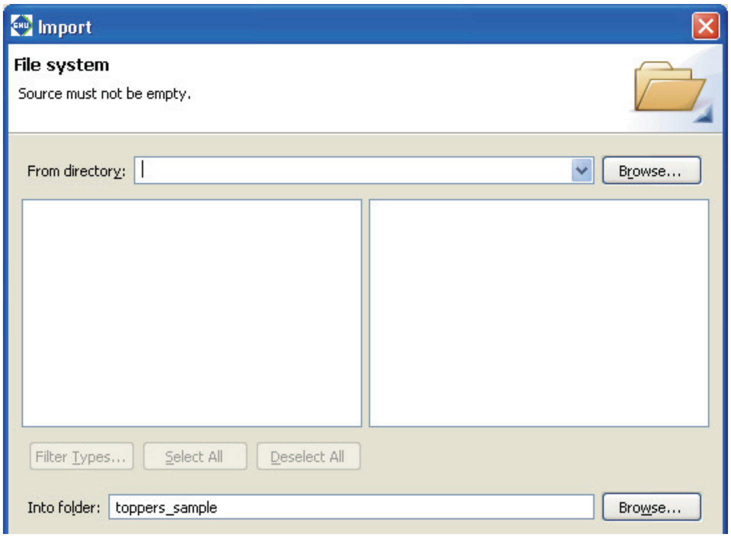
Import the source files after creating a project. Here, for the sake of convenience, we'll also import the makefiles stored in the same directory.

Step 7: Select "`toppers_sample`" in [Navigator] view, and [Import...] from the [File] menu.

This launches the [Import] wizard.


Import
Select
Choose import source.
Select an import source:
type filter text
General
Archive File
Breakpoints
Existing Projects into Workspace
File System
Preferences
C/C++
CVS
Team
< Back
Next >
Finish
Cancel

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

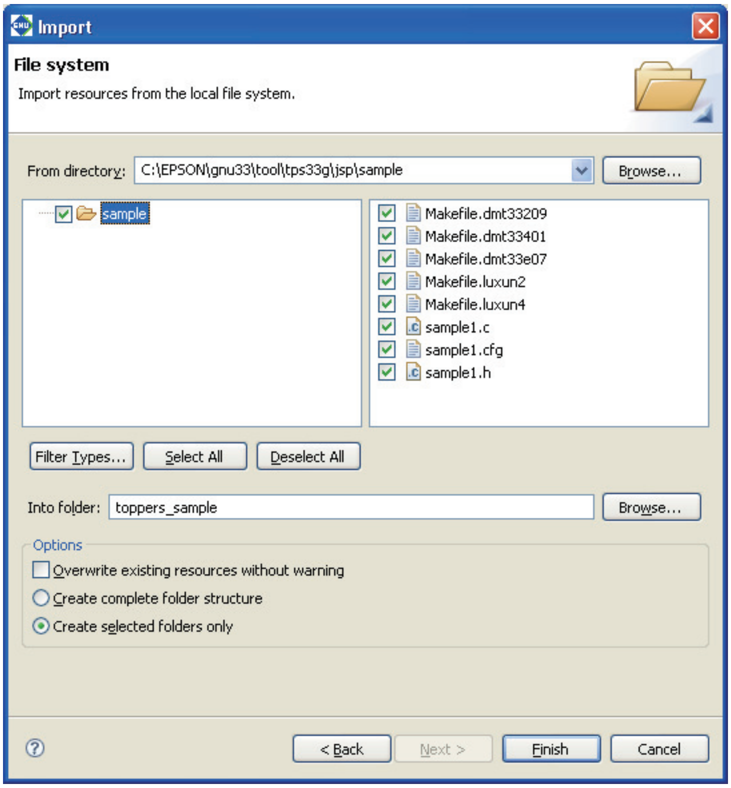

Import
File system
Source must not be empty.
From directory:
Browse...
Filter Types...
Select All
Deselect All
Into folder:
toppers_sample
Browse...

Step 9: Using the [Browse...] button in [From directory:], select the `C:\EPSON\gnu33\tool\tps33g\jsp\sample` directory that contains the files to be imported.

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 box for the `sample` directory shown in the left-side list box.

All of the files in the right-side list will be selected.



Import

File system

Import resources from the local file system.

From directory: C:\EPSON\gnu33\tool\tps33g\jsp\sample

☑ sample

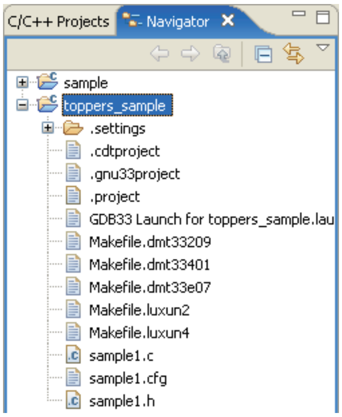
☑ Makefile.dmt33209
☑ Makefile.dmt33401
☑ Makefile.dmt33e07
☑ Makefile.luxun2
☑ Makefile.luxun4
☑ sample1.c
☑ sample1.cfg
☑ sample1.h

Into folder: toppers_sample

Options
☐ Overwrite existing resources without warning
○ Create complete folder structure
● Create selected folders only

Step 11: Click the [Finish] button.

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



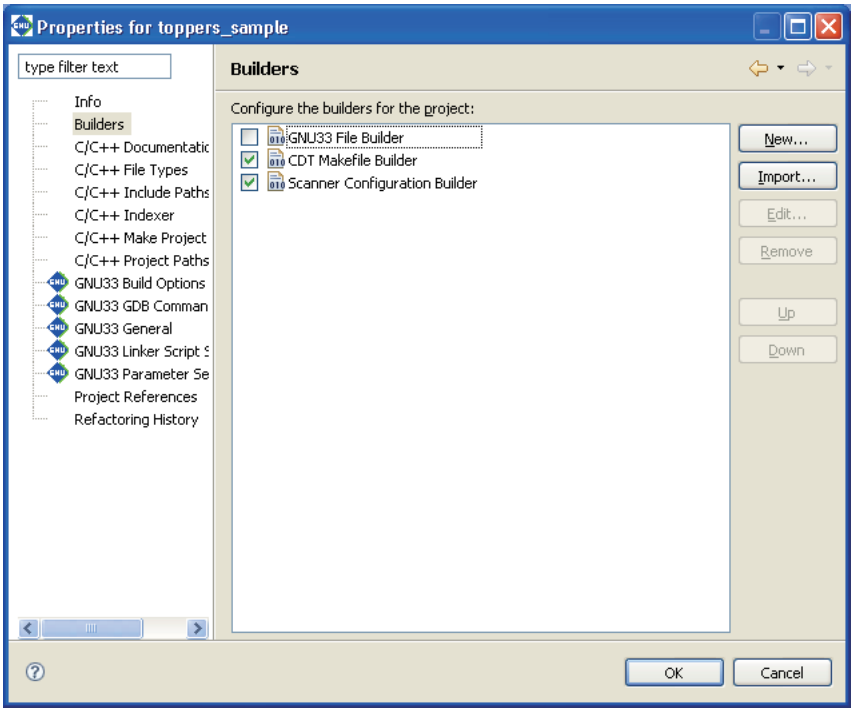
```
⊞ sample
⊟ toppers_sample
   ⊞ .settings
     .cdtproject
     .gnu33project
     .project
     GDB33 Launch for toppers_sample.lau
     Makefile.dmt33209
     Makefile.dmt33401
     Makefile.dmt33e07
     Makefile.luxun2
     Makefile.luxun4
     sample1.c
     sample1.cfg
     sample1.h
```

3.4.3 Disabling the GNU33 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 generating and using these files.
The following describes how to disable the file builder to prevent automatic generation of these files.

Step 12: After selecting the "`toppers_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 [GNU33 File Builder] check box.



Properties for toppers_sample

Builders

Configure the builders for the project:

☐ GNU33 File Builder
☑ CDT Makefile Builder
☑ Scanner Configuration Builder

type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
New...
Import...
Edit...
Remove
Up
Down
OK
Cancel

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**.

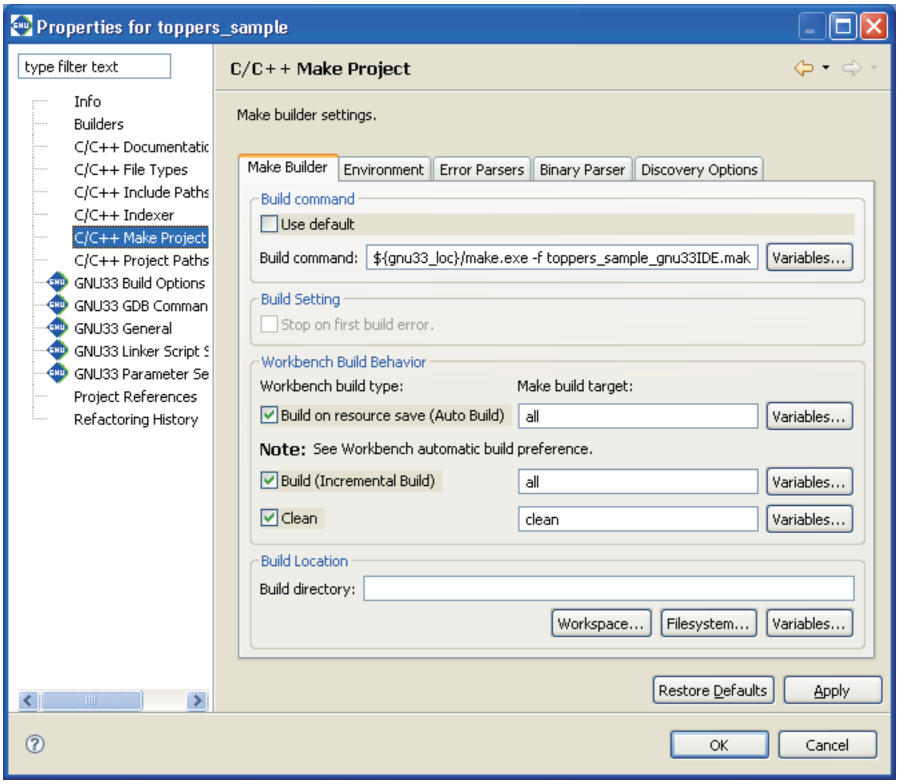
3.4.4 Setting and Correcting the Makefile

To specify a user makefile

Now we'll set up the **IDE** to build the project using the separately prepared makefile. Here, we use `Makefile.dmt33209`, which we imported into the project.

Step 15: After selecting the "`toppers_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.



Properties for toppers_sample

C/C++ Make Project

Make builder settings.

Build command

☐ Use default

Build command: ${gnu33_loc}/make.exe -f toppers_sample_gnu33IDE.mak

Build Setting

☐ Stop on first build error.

Workbench Build Behavior

Workbench build type: Make build target:

☑ Build on resource save (Auto Build) all

Note: See Workbench automatic build preference.

☑ Build (Incremental Build) all

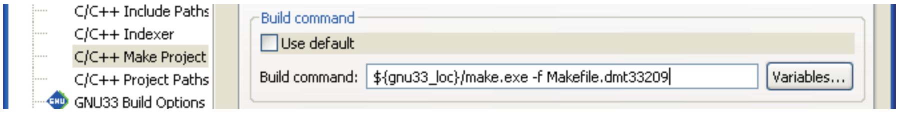
☑ Clean clean

Build Location

Build directory:

type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
Make Builder
Environment
Error Parsers
Binary Parser
Discovery Options
Variables...
Workspace...
Filesystem...
Restore Defaults
Apply
OK
Cancel

Step 17: Change the makefile name "`toppers_sample_gnu33IDE.mak`" set in [Build command:] to "`Makefile.dmt33209`".



Build command

☐ Use default

Build command: ${gnu33_loc}/make.exe -f Makefile.dmt33209

C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
Variables...

No change is required if the makefile is `Makefile.dmt33209`. 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.

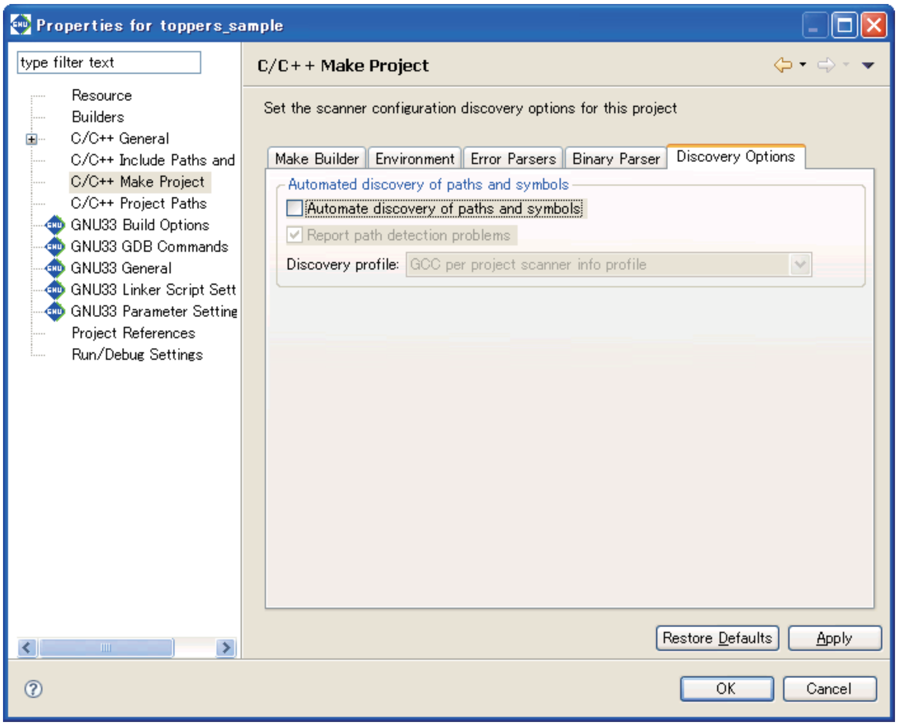
[Clean]

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: Display the page for the [Discovery Options] tab.

Step 19: Deselect the [Automate discovery of paths and symbols] check box.


Properties for toppers_sample
type filter text
Resource
Builders
C/C++ General
C/C++ Include Paths and
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Commands
GNU33 General
GNU33 Linker Script Sett
GNU33 Parameter Setting
Project References
Run/Debug Settings
C/C++ Make Project
Make Builder
Environment
Error Parsers
Binary Parser
Discovery Options
Automated discovery of paths and symbols
Restore Defaults
Apply
OK
Cancel

Set the scanner configuration discovery options for this project

☐ Automate discovery of paths and symbols

☑ Report path detection problems

Discovery profile: GCC per project scanner info profile

Step 20: Click the [OK] button.

Correcting the makefile contents

To use a makefile created for the old version, you may need to change the path written in it. Since the source directory of `Makefile.dmt33209` is indicated by 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.

Step 21: Double-click on the file name "`Makefile.dmt33209`" displayed in [Navigator] view to open it in the editor.

Step 22: As shown below, change the defined content of the macro "`SRCDIR`" indicating the path to the source directory.
Before change: `SRCDIR=      ..`
After change: **`SRCDIR=/cygdrive/c/EPSON/gnu33/tool/tps33g/jsp`**

* This correction is not necessary if you set the project directory to `C:\EPSON\gnu33\tool\tps33g\jsp\sample` when creating the new project.
If you installed `tps33g` in another directory, change the path to that directory.

Step 23: To save the file, click the [Save] button in the toolbar, or select [Save] from the [File] menu.

Step 24: Use the close (☒) button on the editor tab to close the file.

Steps 25 through 28 are mandatory if the gnu33 tools are installed in a directory other than `C:\EPSON\gnu33`. The steps are not necessary if the gnu33 tools are installed in the `C:\EPSON\gnu33` directory.

The makefiles supplied in the Toppers sample such as `Makefile.dmt33209` include a file named "`Makefile.config`" due to the `include` directive.

```
include $(SRCDIR)/config/$(CPU)-$(TOOL)/Makefile.config
```

This file contains a macro specifying the gnu33 tool directory. If the gnu33 tools are installed in another directory, you must revise the macro definition in this file.

Step 25: Select [Open File...] from the [File] menu. From the ensuing file select dialog box, select `Makefile.config` in the `\tps33g\jsp\config\s1c33-gnu33\dmt33209` directory and open in the editor.

Step 26: Change the specified content of the "`TOPDIR`" macro to the directory where the gnu33 tools are installed.
Before change: `TOPDIR = /cygdrive/c/EPSON/gnu33`
After change: **`TOPDIR = /cygdrive/d/EPSON/gnu33`**
(If installed in `D:\EPSON\gnu33`)

Step 27: To save the file, click the [Save] button in the toolbar, or select [Save] from the [File] menu.

Step 28: Use the close (☒) button on the editor tab to close the file.

Although this tutorial does not require these corrections, the following are various locations within the makefile that require correction, as well as a sample linker script file if the linker script file needs to be changed to one other than the sample, or if its content require correction.

- Macro-definitions that specify a linker script file in the makefile
  File: `Makefile.config` in the `C:\EPSON\gnu33\tool\tps33g\jsp\config\s1c33-gnu33\dmt33209` directory
  Macro-definition: `LDSCRIPT = $(CPU)/$(SYS)/dmt33209.ld`

  Alter the `LDSCRIPT` definition to use other makefiles.

- Linker script file used in `Makefile.dmt33209`
  `C:\EPSON\gnu33\tool\tps33g\jsp\config\s1c33\dmt33209\dmt33209.ld`

  To revise the contents of the linker script file, open this file and make the necessary changes.

(Although this tutorial uses DMT33209, directories for boards other than `\dmt33209` are available, if needed.)

To correct a file, use [Open File...] to open the file. Make the corrections in the same way as in Step 25.

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 29: Select the project "`toppers_sample`" name from the [Navigator] or the [C/C++ Projects] view.

Step 30: Select [Build Project] from the [Project] menu.

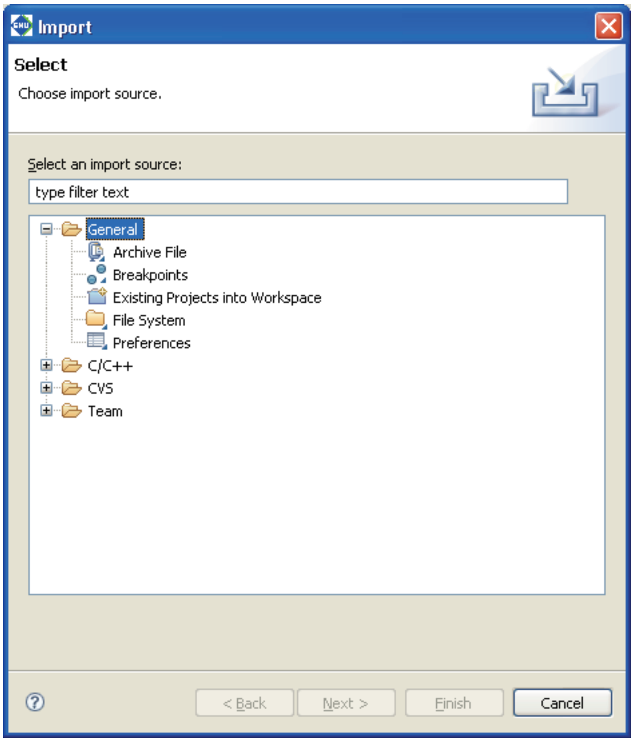
When the build command is selected this way, **make.exe** is executed with a specified makefile to generate the executable format object file "`sample1.elf`". (Since we are using a makefile that is not automatically generated, the object file is not named after the project name.)

3.4.6 Importing Debugger Startup Files

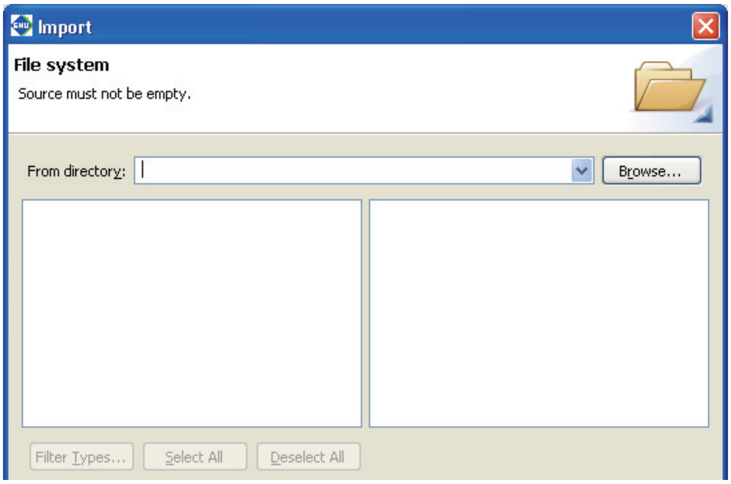
User-prepared files can be used for the command and parameter files used by the debugger at startup. Here, we'll use the `icdv3.cmd` (command file) and `dmt33209.par` (parameter file) found in the directory `C:\EPSON\gnu33\tool\tps33g\jsp\config\s1c33-gnu33\dmt33209`. (Although this tutorial uses the DMT33209, directories and files for boards other than `\dmt33209` are available, if needed.)
First, import these files into the project.

Step 31: Select the "`toppers_sample`" project from the [Navigator] view, then [Import...] from the [File] menu.

This launches the [Import] wizard.


Import
Select
Choose import source.
Select an import source:
type filter text
General
Archive File
Breakpoints
Existing Projects into Workspace
File System
Preferences
C/C++
CVS
Team
< Back
Next >
Finish
Cancel

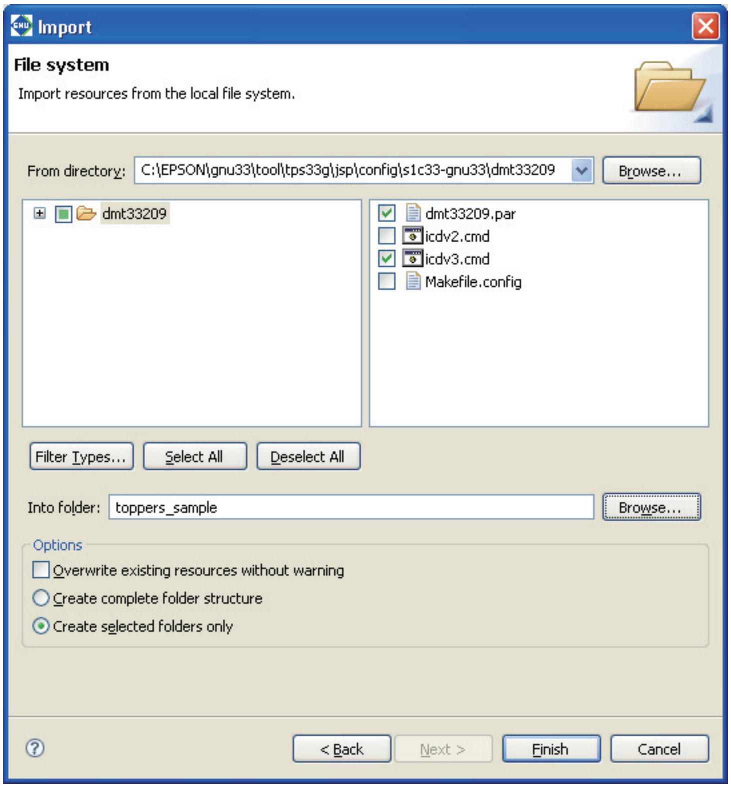
Step 32: Select [General] > [File System] from the displayed list and click the [Next>] button.


Import
File system
Source must not be empty.
From directory:
Browse...
Filter Types...
Select All
Deselect All

Step 33: Using the [Browse...] button in [From directory:], select the directory `C:\EPSON\gnu33\tool\tps33g\jsp\config\s1c33-gnu33\dmt33209`, which contains the files to be imported.

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 34: Select the check boxes "`dmt33209.par`" and "`icdv3.cmd`" shown in the list box to the right.



Import

File system

Import resources from the local file system.

From directory: C:\EPSON\gnu33\tool\tps33g\jsp\config\s1c33-gnu33\dmt33209

- dmt33209

☑ dmt33209.par
☐ icdv2.cmd
☑ icdv3.cmd
☐ Makefile.config

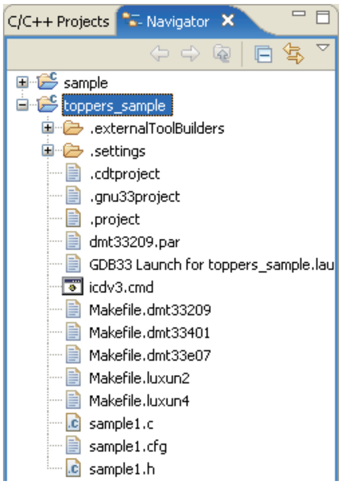
Into folder: toppers_sample

Options

☐ Overwrite existing resources without warning
○ Create complete folder structure
● Create selected folders only

Step 35: Click the [Finish] button.

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



- sample
- toppers_sample
  - .externalToolBuilders
  - .settings
  - .cdtproject
  - .gnu33project
  - .project
  - dmt33209.par
  - GDB33 Launch for toppers_sample.lau
  - icdv3.cmd
  - Makefile.dmt33209
  - Makefile.dmt33401
  - Makefile.dmt33e07
  - Makefile.luxun2
  - Makefile.luxun4
  - sample1.c
  - sample1.cfg
  - sample1.h

3.4.7 Correcting Debugger Startup Files

If the command and parameter files created for use with the old version are used, the paths specified in these files may need to be changed. For `icdv3.cmd`, for example, since the flash program to be used is specified by a relative path, the specified path must be corrected to a cygwin format path to allow file referencing from the project directory.

Step 36: Double-click on the file name "`icdv3.cmd`" displayed in [Navigator] view to open it in the editor.

Step 37: Change the `file` command, as shown below.
Before change: `file ../config/s1c33-gnu33/dmt33209/fls/am29f800.elf`
After change: **`file /cygdrive/c/EPSON/gnu33/tool/tps33g/jsp/config/s1c33-gnu33/dmt33209/fls/am29f800.elf`**

* This correction is not required if you set the project directory to `C:\EPSON\gnu33\tool\tps33g\jsp\sample` when creating a new project.
If you installed `tps33g` in another directory, change the path to the appropriate directory.

Step 38: Include any commands that need to be added in the file. (This is not required by this tutorial.)

Step 39: To save the file, click the [Save] button in the toolbar, or select [Save] from the [File] menu.

Step 40: Use the close (☒) button on the editor tab to close the file.

This command file executes the `c33 rpf` command (to set a memory map) by specifying the parameter file `dmt33209.par` stored in the same directory. Although not required in this tutorial, if the parameter file needs to be changed to other than the sample, revise the "`c33 rpf dmt33209.par`" command line. Additionally, open `dmt33209.par` and edit it if map information in the sample parameter file needs to be revised.

3.4.8 Starting the Debugger

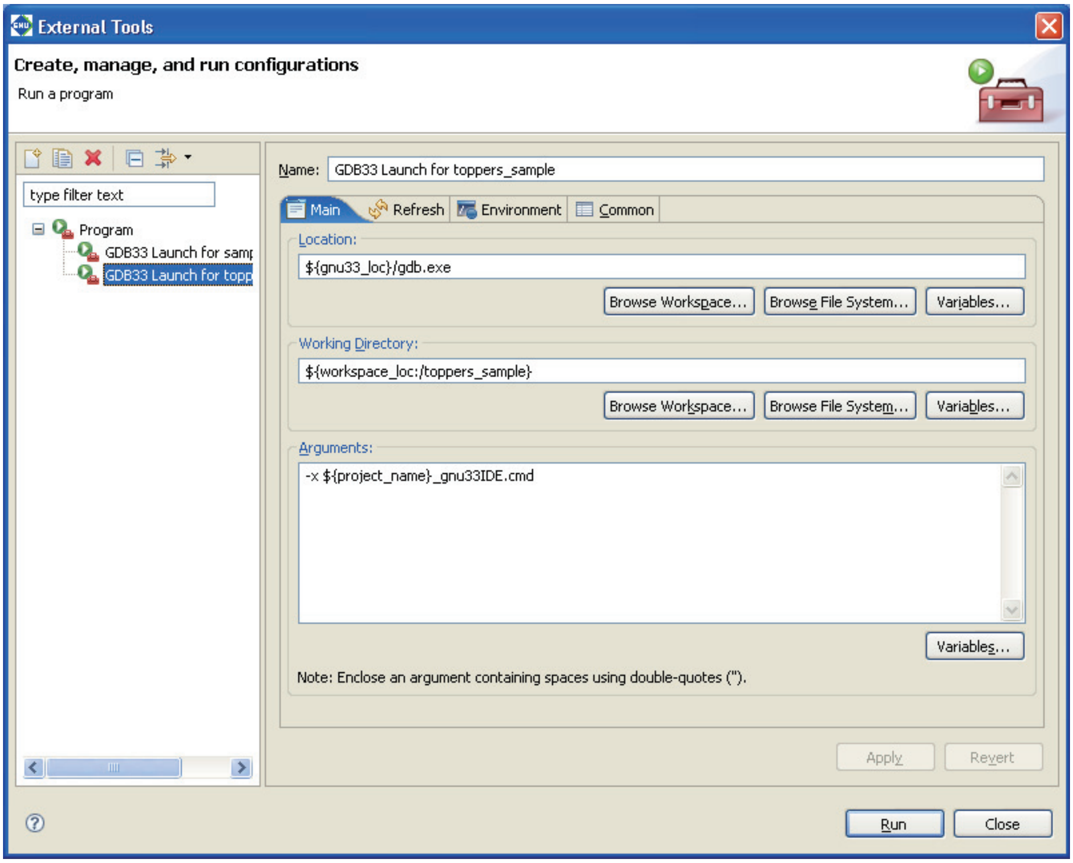
As the last step in Tutorial 2, described below is the method for making the necessary settings to execute the prepared command file at debugger startup.

If you want to execute the sample program after configuring these settings, please make sure the ICD (S5U1C33001H) and demonstration board are connected and powered on so that you can begin debugging. For more information on connections with the ICD, board settings, operations, etc., refer to `readme_s1c33.txt` (in English) or `readme_ja_s1c33.txt` (in Japanese). The files are found in the `C:\EPSON\gnu33\tool\tps33g` directory.

Step 41: Select [External Tools] > [External Tools Configurations...] from the [Run] menu.

The [External Tools Configurations] dialog box will be displayed.

Step 42: Select [GDB33 Launch for toppers_sample] from the list of the dialog box and display the page for the [Main] tab.



External Tools

Create, manage, and run configurations

Run a program

type filter text

- Program
  - GDB33 Launch for samp
  - GDB33 Launch for topp

Name: GDB33 Launch for toppers_sample

Location: ${gnu33_loc}/gdb.exe

Working Directory: ${workspace_loc:/toppers_sample}

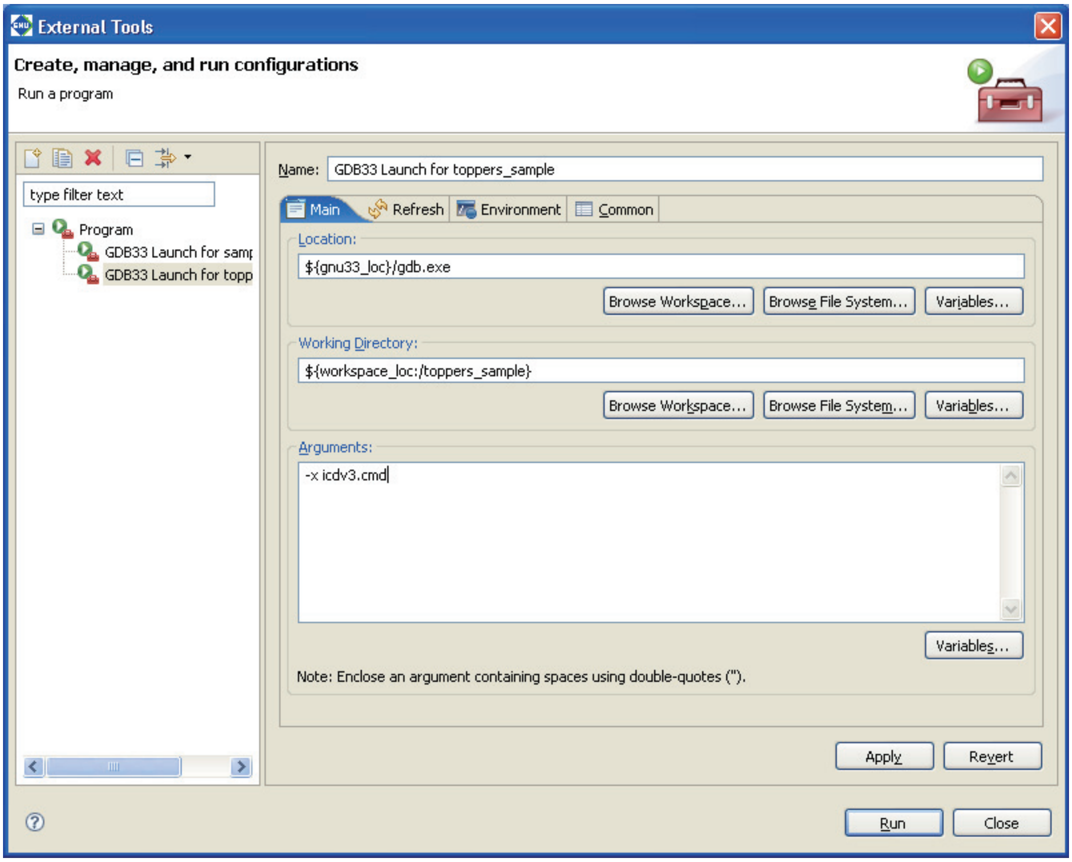
Arguments: -x ${project_name}_gnu33IDE.cmd

Note: Enclose an argument containing spaces using double-quotes (").

Step 43: In the [Arguments:] column, modify the `-x` option that specifies a command file as shown below.

Before change: `-x ${project_name}_gnu33IDE.cmd`

After change: `-x icdv3.cmd`


External Tools
Create, manage, and run configurations
Run a program
type filter text
Program
GDB33 Launch for samp
GDB33 Launch for topp
Main
Refresh
Environment
Common
Browse Workspace...
Browse File System...
Variables...
Browse Workspace...
Browse File System...
Variables...
Variables...
Apply
Revert
Run
Close

Name: GDB33 Launch for toppers_sample

Location: ${gnu33_loc}/gdb.exe

Working Directory: ${workspace_loc:/toppers_sample}

Arguments: -x icdv3.cmd

Note: Enclose an argument containing spaces using double-quotes (").

Step 44: Click the [Apply] button to confirm what you've altered here.

Starting the debugger will now execute `icdv3.cmd`.

Step 45: 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 S1C33 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\gnu33\sample_ide\std\simulator\ansilib`

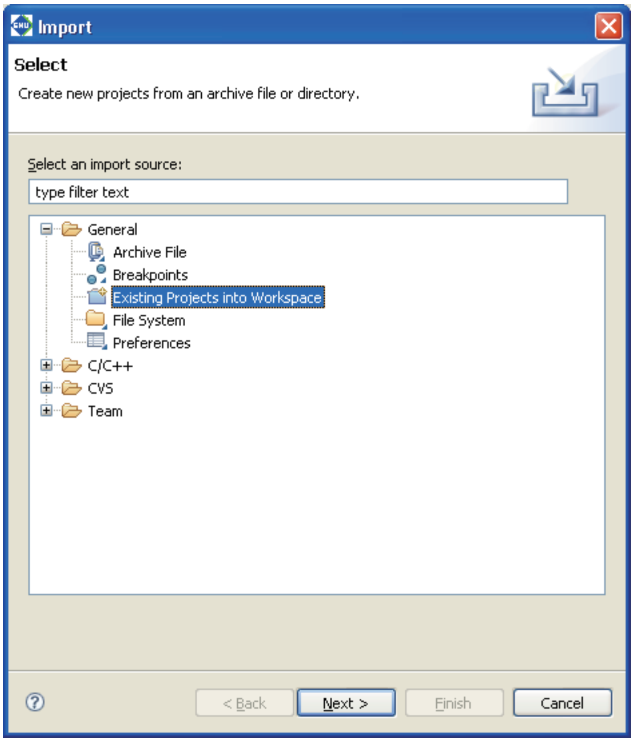
To import a project

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.



Select

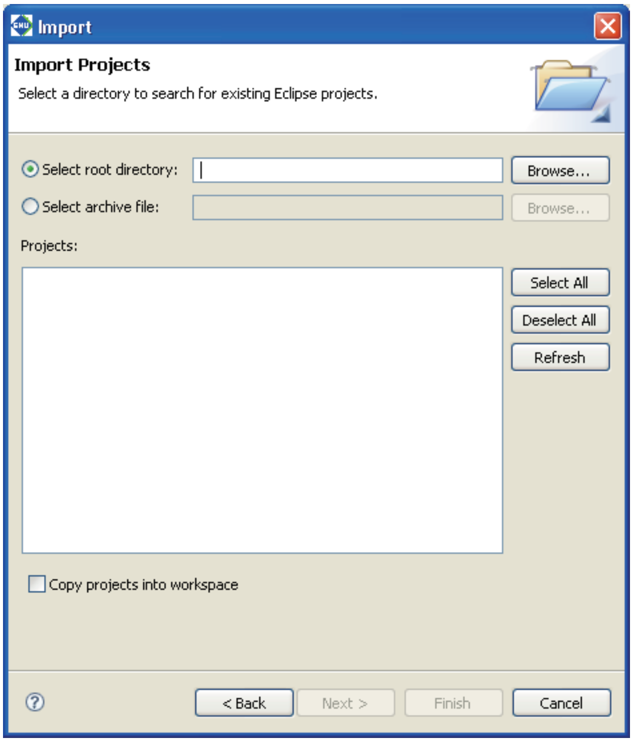
Create new projects from an archive file or directory.

Select an import source:

type filter text

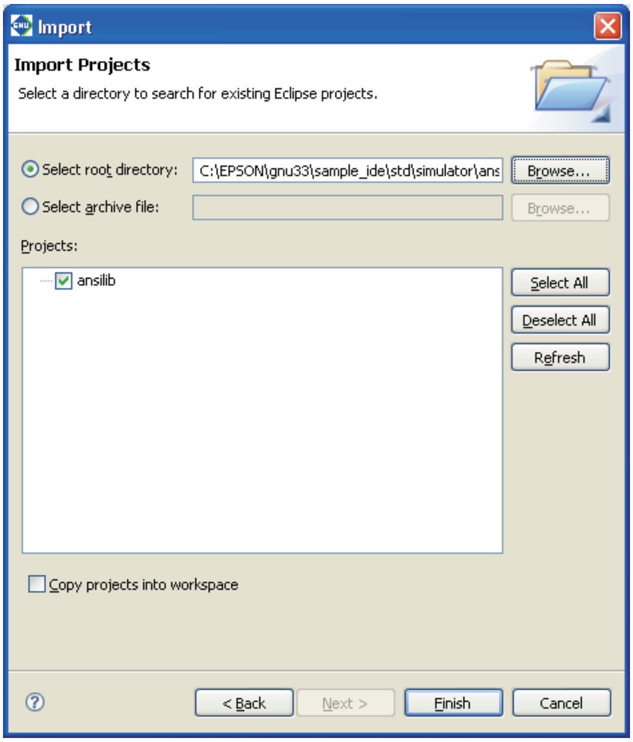
- General
  - Archive File
  - Breakpoints
  - Existing Projects into Workspace
  - File System
  - Preferences
- C/C++
- CVS
- Team

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



Import
Import Projects
Select a directory to search for existing Eclipse projects.
○ Select root directory:
○ Select archive file:
Browse...
Browse...
Projects:
Select All
Deselect All
Refresh
☐ Copy projects into workspace
< Back
Next >
Finish
Cancel

Step 4: Using the [Browse...] button for [Select root directory:], select the project directory `C:\EPSON\gnu33\sample_ide\std\simulator\ansilib` to be imported.



Import
Import Projects
Select a directory to search for existing Eclipse projects.
Browse...
Browse...
Projects:
Select All
Deselect All
Refresh
< Back
Next >
Finish
Cancel

● Select root directory: C:\EPSON\gnu33\sample_ide\std\simulator\ans

○ Select archive file:

☑ ansilib

☐ Copy projects into workspace

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.

* Do not specify the project directory (directory containing .project file) as a workspace directory. Doing so may result in failures with project imports (when [Copy projects into workspace] is selected).
The current workspace directory can be checked by selecting [File] > [Switch workspace...] > [Others...] and opening the [Workspace Launcher] dialog box.

Step 6: Click the [Finish] button.

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

When a project is imported this way, the project directory is not copied to the workspace. Work is performed directly at the position selected when importing a project. If the selected project directory is the original project directory, make a backup copy before importing, since the following steps will modify its content. Otherwise, make a copy of the project after importing it according to the procedure described below.

To make a copy of a project

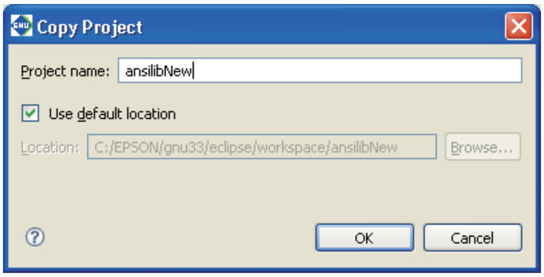
Step 7: In the [C/C++ Projects] view, select the imported project "`ansilib`".

Step 8: Select [Copy] from the [Edit] menu (or Ctrl + C), then [Paste] from the [Edit] menu (or Ctrl + V).

The [Copy Project] dialog box will be displayed.

Step 9: Enter a project name (in this case, "`ansilibNew`") in the [Project name:] field.

Step 10: Select the [Use default location] check box.



Project name: ansilibNew

☑ Use default location

Location: C:/EPSON/gnu33/eclipse/workspace/ansilibNew

When this check box is selected, the project will be copied into the workspace under the name "`ansilibNew`".
If you want to copy a project to somewhere other than the workspace, enter a path to the desired location in the [Location:] field, or use the [Browse...] button and select one from the displayed list.

Step 11: Click the [OK] button to close the dialog box.

The project named "`ansilibNew`" is created in the [C/C++ Projects] view. The directory for this project is newly created in the workspace.

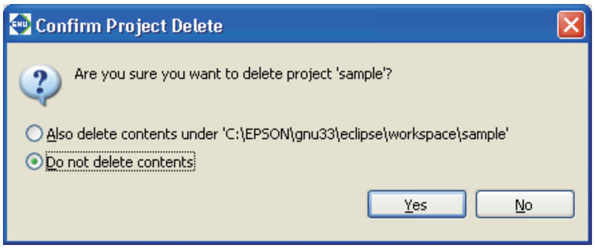
The following shows how to delete the original "`ansilib`" project from the [C/C++ Projects] view.

Deleting a project

Step 12: In the [C/C++ Projects] view, select the project "`ansilib`".

Step 13: Press the [Delete] key.

This displays the [Confirm Project Delete] dialog box.



Are you sure you want to delete project 'sample'?

○ Also delete contents under 'C:\EPSON\gnu33\eclipse\workspace\sample'

● Do not delete contents

Step 14: Click [Yes] after confirming that the option [Do not delete contents] is selected.

The "`ansilib`" project will be deleted from the [C/C++ Projects] view.
If the option [Do not delete contents] is selected, the directory for the project in the file system is not deleted. Keep in mind that if the option [Also delete contents under .....] is selected, the contents of the directory in the file system will be deleted as well.

Automatic updates of the project file

When you import a project created in an older version of the IDE, the project file (.project/.cproject/.gnu33project/) is automatically updated to one compatible with the current version.
Note that .cdtproject files used by projects of an older version will be replaced by .cproject files. A .cproject file is generated during project import, and the contents of the .cdtproject file will be transferred automatically to the newly generated .cproject file.

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\gnu33`). Otherwise, corrections are required for the user library and include directory. Make these corrections in the [GNU33 Build Options] of the [Properties] dialog box for the project.

3.6 Debugging Environment

The debugger supports the S5U1C330M2S debug monitor and the S5U1C33000H and S5U1C33001H in-circuit debuggers, allowing you to debug a program by actually operating the target system.

The respective debugging systems and program debugging are briefly discussed below. For more information, refer to the manuals provided with each debugging system. If you need information on on-board flash programming, refer to `Readme.txt` (in English) or `Readmeja.txt` (in Japanese). These files are found in `\gnu33\utility\fls33`.

Notes:
- Before connecting and disconnecting equipment in a debugging system, always confirm that power for all equipment has been switched off.
- The sample program used in this section is located in the `\EPSON\gnu33\utitily\sample_std` directory. Although executing the program is possible in its current directory, move the `\sample_std` directory under `\gnu33` in order to rebuild it with **make.exe**.

3.6.1 Debug Monitor S5U1C330M2S

Middleware for the S1C33 Family, the S5U1C330M2S debug monitor allows you to debug a program on the S5U1C33xxxD or user target board. By connecting a board incorporating the S5U1C330M2S to your computer via the S5U1C330M1D1 board, you can use the **gdb** debugger to debug a program. Discussed here is the procedure for debugging a program using the S5U1C33xxxD board incorporating the debug monitor as a development tool.

System configuration and connections

Figure 3.6.1.1 shows the configuration of a system for debugging a program using the S5U1C33xxxD board.

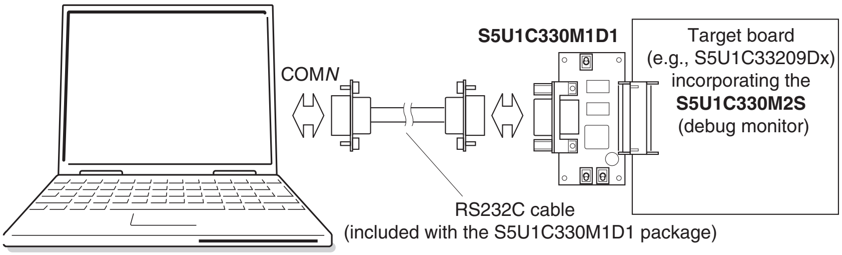


Figure 3.6.1.1 Debug system configuration using S5U1C330M1D1 and S5U1C33xxxD board

Startup and operation verification

The sample programs listed below may be used to verify program operation.

- `\EPSON\gnu33\utility\sample_std\dmt33209\`**`led.elf`**
- `\EPSON\gnu33\utility\sample_std\dmt33209\`**`led2.elf`**

These sample programs may be used for all S5U1C33xxxD boards. They are designed to illuminate the LEDs on the board. The programs `led.elf` and `led2.elf` are created to be executed in RAM (0x600000 and over) and in flash memory (0x200000 and over), respectively.

For the contents of each program, refer to the source file (`led.s`) stored in the directory. Executable format object files are included as samples. Unless source corrections are required, you do not need to run **make.exe.** If the source of either program needs to be corrected for debugging, run **make.exe** using the makefile for that program.

(1) Starting the debug monitor

The boot routine mapped to memory beginning with S5U1C33xxxD address 0xc00000 is designed so that the debug monitor is launched when the K63 input port is set to 0 ([DEBUG] switch on S5U1C330M1D1 turned on). After connecting the target system and your computer, start the debug monitor as described below.

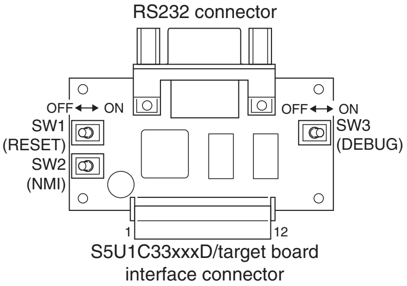


Figure 3.6.1.2 S5U1C330M1D1 board layout

1. SW3 [DEBUG] on S5U1C330M1D1 on.
2. Turn power on for the S5U1C33xxxD.
3. Reset the S5U1C33xxxD (by turning SW6 [RESET] on S5U1C33xxxD off, if on).
4. Turn on power to the computer to start Windows.
5. Launch the **gdb** debugger (this is described further below).

Note: If you switch on power to the S5U1C33xxxD on while SW3 [DEBUG] on S5U1C330M1D1 is off, the S5U1C33xxxD starts executing the program from the boot address at the beginning of flash memory (0x200000 and over) without starting up the debug monitor. In such cases, turn SW3 [DEBUG] on and use SW6 [RESET] on S5U1C33xxxD to reset it before starting the debug monitor.

(2) Debugging a program in RAM

The sample program for debugging in S5U1C33xxxD RAM (0x600000 and over) is `led.elf`. When starting the debugger, specify in the `-x` option the debug command file "`icdv2_led.cmd`", which sets a vector table address at the beginning of the RAM and loads `led.elf` into the RAM. The procedure for starting the debugger from the command prompt is described below.

1. Start the debugger as described above.
2. Set `\EPSON\gnu33\utility\sample_std\dmt33209` to the current directory.
3. Set the path to **gdb.exe**.
4. Run the command shown below from the command prompt to start the debugger.

```
C:\EPSON\gnu33\utility\sample_std\dmt33209\>gdb -x icdv2_led.cmd
```

The debugger is set in debug monitor mode after startup, allowing the `led.elf` to be debugged. For example, when you execute the `continue` command, the LED on the S5U1C33xxxD will flash.

In the debug monitor you cannot use forced break facilities such as key breaks.
For this reason, `icdv2_led.cmd` has a breakpoint set at a label position in the NMI routine of `led.elf`. When you turn on SW5 [NMI] on S5U1C33xxxD, a NMI interrupt is generated, allowing you to forcibly break program execution.

(3) Debugging a program in flash memory

The sample program to be used for debugging in the flash memory of the S5U1C33xxxD (0x200000 and over) is `led2.elf`.

To write the sample program into flash memory, first load the flash erase/programming routine "`am29f800.elf`" into the debugger.

Then execute the `c33 fls` and the `c33 fle` commands to initialize flash-related settings and erase the flash memory. Execute the `load` command to load the sample program. For more information on using these commands, refer to the debug command file "`icdv2_led2.cmd`" included in the sample.

The debug command file "`icdv2_led2.cmd`" includes debug commands that load the flash erase/programming routine, set a vector table address, and write `led2.elf` into flash memory. Specify it in the `-x` option when you start the debugger.

The procedure for starting the debugger from the command prompt is described below.

1. Start the debugger as described above.
2. Set `\EPSON\gnu33\utility\sample_std\dmt33209` to the current directory.
3. Set the path to **gdb.exe**.
4. Execute the command shown below from the command prompt to start the debugger.

```
C:\EPSON\gnu33\utility\sample_std\dmt33209\>gdb -x icdv2_led2.cmd
```

The debugger is set in debug monitor mode after startup, allowing the `led2.elf` to be debugged. For example, when you execute the `continue` command, the LED on the S5U1C33xxxD board will flash.

In the debug monitor you cannot use forced break facilities such as key breaks.
When you turn on SW5 [NMI] on S5U1C33xxxD, a NMI interrupt is generated, allowing you to forcibly break program execution.

The program written into flash memory can be executed singly with the S5U1C33xxxD alone.
After quitting the debugger, turn the power to the system off, disconnect RS232C cable, and turn SW3 [DEBUG] on S5U1C330M1D1 off. Then turn the power to the S5U1C33xxxD on. This will cause `led2.elf` to be executed in the flash memory, with the LED made to blink.

Precautions

When a user program is debugged using the S5U1C33xxxD, keep in mind the following points.

(1) The debug monitor for the S5U1C33xxxD is implemented by linking `mon33ch0.lib`. Therefore, the internal serial interface Ch.0 cannot be used from a user program.

(2) In the debug monitor, forced break facilities such as key breaks cannot be used.
If forced break facilities are needed, set labels in NMI or key-input interrupt handler routines of the program to be debugged, then set hardware PC breaks at these labels.

(3) Programs are downloaded into RAM at approximately 8KB/s, while programs are downloaded into flash memory is approximately 7KB/s. Note that these speeds will vary with the PC used and operating conditions.

(4) Create the program to be debugged so that it can be loaded into a free area in the RAM or flash memory of the S5U1C33xxxD before execution. Since the address to which a program is loaded cannot be specified from the debugger, make sure this address is determined when the program is linked.
In particular, keep in mind that the addresses 0–0x2f in internal RAM and the addresses 0x6ff640–0x6fffff in external RAM are used by the S5U1C330M2S. If these locations are rewritten, the debug monitor will not function.
When memory contents are rewritten with a memory-manipulating command, make sure these locations are not altered.

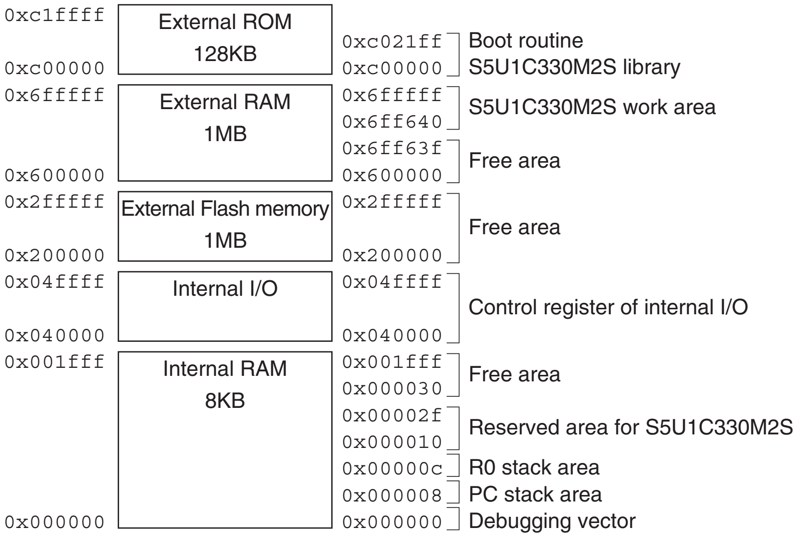

0xc1ffff
External ROM
128KB
0xc00000
0xc021ff
0xc00000
Boot routine
S5U1C330M2S library
0x6fffff
External RAM
1MB
0x600000
0x6fffff
0x6ff640
S5U1C330M2S work area
0x6ff63f
0x600000
Free area
0x2fffff
External Flash memory
1MB
0x200000
0x2fffff
0x200000
Free area
0x04ffff
Internal I/O
0x040000
0x04ffff
0x040000
Control register of internal I/O
0x001fff
Internal RAM
8KB
0x000000
0x001fff
0x000030
Free area
0x00002f
0x000010
Reserved area for S5U1C330M2S
0x00000c
R0 stack area
0x000008
PC stack area
0x000000
Debugging vector

Figure 3.6.1.3 Memory map of the S5U1C33xxxD

(5) For other limitations on use of the debug monitor, refer to the document entitled "S1C33 Family Debug Monitor Operation Manual".

3.6.2 In-Circuit Debugger S5U1C33000H

The in-circuit debugger S5U1C33000H controls the debug facilities of the S1C33 chip according to the **gdb** debugger commands. In addition to the debug facilities equivalent to these of the debug monitor, it offers a trace facility realized through internal trace memory. Discussed here is the method for debugging a program using the S5U1C33000H and the S5U1C33xxxD board as development tools.

System configuration and connections

Figure 3.6.2.1 shows the configuration of a system for debugging a program using the S5U1C33000H and the S5U1C33xxxD board.

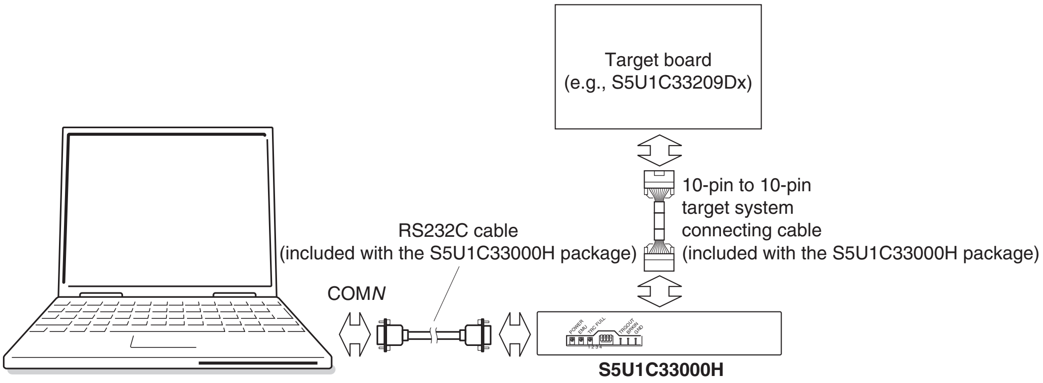


Figure 3.6.2.1 Debug system configuration using S5U1C33000H and S5U1C33xxxD board

Note: The S5U1C33000H and the debug monitor cannot be used in combination. Do not connect the S5U1C330M1D1 to the S5U1C33xxxD board. If the S5U1C330M1D1 needs to be attached, make sure that SW3 [DEBUG] is turned off.

Startup and operation verification

The sample programs listed below may be used to verify program operation.

- `\EPSON\gnu33\utility\sample_std\dmt33209\`**`led.elf`**
- `\EPSON\gnu33\utility\sample_std\dmt33209\`**`led2.elf`**

These sample programs may be used for all S5U1C33xxxD boards. They are designed to illuminate the LEDs on the board. The programs `led.elf` and `led2.elf` are created to be executed in RAM (0x600000 and over) and in flash memory (0x200000 and over), respectively.

For the contents of each program, refer to the source file (`led.s`) stored in the directory. Executable format object files are included as samples. Unless source corrections are required, you do not need to run **make.exe.** If the source of either program needs to be corrected for debugging, run **make.exe** using the makefile for that program.

(1) Starting the system

After connecting the S5U1C33000H, the S5U1C33xxxD, and a PC, start the system, as described below.

1. Leave all DIP switches of the S5U1C33000H open (by pushing them up).
2. Turn on power for the S5U1C33000H.
3. Turn on power for the S5U1C33xxxD.
4. Turn on power for the PC to start Windows.
5. Start the **gdb** debugger (this is described further below).

(2) Debugging a program in RAM

The sample program to be used for debugging in the RAM of the S5U1C33xxxD (0x600000 and over) is `led.elf`. A debug command file "`icdv2_led.cmd`" that sets a vector table address at the beginning of the RAM and loads `led.elf` into the RAM is available. Specify it in the `-x` option when you start the debugger.

The procedure for starting the debugger from the command prompt is described below.

1. Start the system as described above.
2. Set `\EPSON\gnu33\utility\sample_std\dmt33209` to the current directory.
3. Set the path to **gdb.exe**.
4. Execute the command shown below from the command prompt to start the debugger.

```
C:\EPSON\gnu33\utility\sample_std\dmt33209\>gdb -x icdv2_led.cmd
```

The debugger is set in ICD2 mode after startup, allowing the `led.elf` to be debugged. For example, when you execute the `continue` command, the LED on the S5U1C33xxxD board will flash.

In the S5U1C33000H, you can use a key break facility to break program execution using the debugger [Stop] button. You also can use a trace facility. For more information on trace, refer to Chapter 10, "Debugger".

(3) Debugging a program in flash memory

The sample program to be used for debugging in the flash memory of the S5U1C33xxxD (0x200000 and over) is `led2.elf`.

To write the sample program into flash memory, load the flash erase/programming routine "`am29f800.elf`" into the debugger. Then execute the `c33 fls` and the `c33 fle` commands to initialize flash-related settings and erase the flash memory. Execute the `load` command to load the sample program. For more information on using these commands, refer to the debug command file "`icdv2_led2.cmd`" included in the sample.

The debug command file "`icdv2_led2.cmd`" includes debug commands that load the flash erase/programming routine, set a vector table address, and write `led2.elf` into flash memory. Specify it in the `-x` option when you start the debugger.

The procedure for starting the debugger from the command prompt is described below.

1. Start the debugger as described above.
2. Set `\EPSON\gnu33\utility\sample_std\dmt33209` to the current directory.
3. Set the path to **gdb.exe**.
4. Execute the command shown below from the command prompt to start the debugger.

```
C:\EPSON\gnu33\utility\sample_std\dmt33209\>gdb -x icdv2_led2.cmd
```

Precautions

When a user program is debugged using the S5U1C33000H and S5U1C33xxxD, keep in mind the following points.

(1) The program to be debugged should be created so that it can be loaded into a free area in the RAM or flash memory of the S5U1C33xxxD before execution. Since the address to which a program is loaded cannot be specified from the debugger, make sure this address is determined when the program is linked. For a memory map in the S5U1C33xxxD, refer to Figure 3.6.1.3.

(2) The entire facility of the S5U1C33000H can be made usable only by connecting it to the COM port of the PC with RS232C cable.
Programs are downloaded into RAM is approximately 8KB/s, and programs are downloaded into flash memory is approximately 7KB/s. Note, however, that these speeds vary with the PC used and operating conditions.

(3) For other limitations on use of the S5U1C33000H, refer to the document entitled "S1C33 Family In-Circuit Debugger Manual".

3.6.3 In-Circuit Debugger S5U1C33001H

The in-circuit debugger S5U1C33001H is a tool to control the debug facilities of the S1C33 chip according to the **gdb** debugger commands. In addition to the debug facilities equivalent to these of the debug monitor, it offers a trace facility realized through its internal trace memory. Discussed here is the method for debugging a program using the S5U1C33001H and the S5U1C33xxxD board as development tools.

Notes:
- Supported OS: Windows 2000/XP/Vista
- A dedicated USB driver is required to use the S5U1C33001H for debugging. The USB driver is found in the `\gnu33\utility\drv_usb` directory. For more information on installation, refer to the manual included in the ICD package.

System configuration and connections

Figure 3.6.3.1 shows the configuration of a system for debugging a program using the S5U1C33001H and the S5U1C33xxxD board.

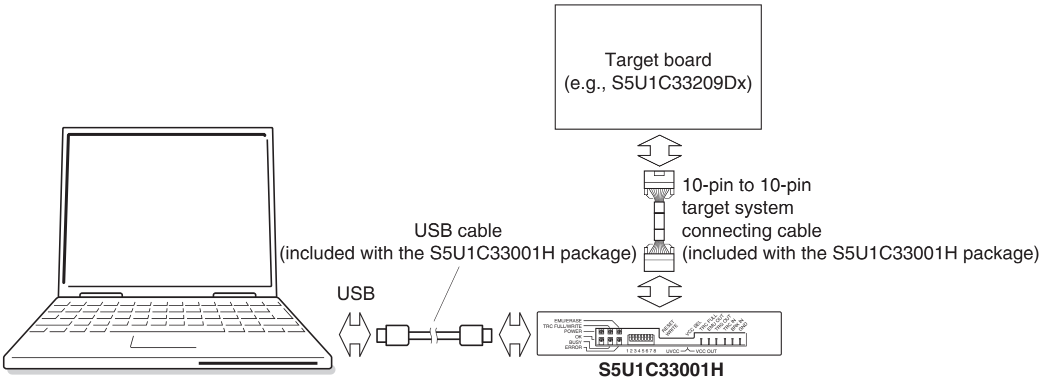

Target board
(e.g., S5U1C33209Dx)
10-pin to 10-pin
target system
connecting cable
(included with the S5U1C33001H package)
USB cable
(included with the S5U1C33001H package)
USB
EMU/ERASE
TRC FULL/WRITE
POWER
OK
BUSY
ERROR
1 2 3 4 5 6 7 8
RESET
WRITE
VCC SEL
TRC FULL
EMU OUT
TRG OUT
TRC IN
BRK IN
GND
UVCC
VCC OUT
S5U1C33001H

Figure 3.6.3.1 Debug system configuration using S5U1C33001H and S5U1C33xxxD board

Note: The S5U1C33001H and the debug monitor cannot be used in combination. Do not connect the S5U1C330M1D1 to the S5U1C33xxxD board. If the S5U1C330M1D1 needs to be attached, always make sure that SW3 [DEBUG] on it is turned off.

Startup and operation verification

The sample programs listed below may be used to verify program operation.

- `\EPSON\gnu33\utility\sample_std\dmt33209\`**`led.elf`**
- `\EPSON\gnu33\utility\sample_std\dmt33209\`**`led2.elf`**

These sample programs may be used for all S5U1C33xxxD boards. They are designed to illuminate the LEDs on the board. The programs `led.elf` and `led2.elf` are created to be executed in RAM (0x600000 and over) and in flash memory (0x200000 and over), respectively.

For the contents of each program, refer to the source file (`led.s`) stored in the directory. Executable format object files are included as samples. Unless source corrections are required, you do not need to run **make.exe.** If the source of either program needs to be corrected for debugging, run **make.exe** using the makefile for that program.

(1) Starting the system

After connecting the S5U1C33001H, the S5U1C33xxxD, and a PC, start the system, as described below.

1. Leave all DIP switches of the S5U1C33001H open (by pushing them up).
2. Turn on power for the S5U1C33001H.
3. Turn on power for the S5U1C33xxxD.
4. Turn on power for the PC to start Windows.
5. Start the **gdb** debugger (this is described further below).

(2) Debugging a program in RAM

The sample program to be used for debugging in the RAM of the S5U1C33xxxD (0x600000 and over) is `led.elf`. A debug command file "`icdv3_led.cmd`" that sets a vector table address at the beginning of the RAM and loads `led.elf` into the RAM is available. Specify it in the `-x` option when you start the debugger.
The procedure for starting the debugger from the command prompt is described below.

1. Start the system as described above.
2. Set `\EPSON\gnu33\utility\sample_std\dmt33209` to the current directory.
3. Set the path to **gdb.exe**.
4. Execute the command shown below from the command prompt to start the debugger.

```
C:\EPSON\gnu33\utility\sample_std\dmt33209\>gdb -x icdv3_led.cmd
```

The debugger is set in ICD3 mode after startup, allowing the `led.elf` to be debugged. For example, when you execute the `continue` command, the LED on the S5U1C33xxxD board will flash.

In the S5U1C33001H, you can use a key break facility to break program execution using the debugger [Stop] button. You also can use a trace facility. For more information on trace, refer to Chapter 10, "Debugger".

(3) Debugging a program in flash memory

The sample program to be used for debugging in the flash memory of the S5U1C33xxxD (0x200000 and over) is `led2.elf`.
To write the sample program into flash memory, load the flash erase/programming routine "`am29f800.elf`" into the debugger. Then execute the `c33 fls` and the `c33 fle` commands to initialize flash-related settings and erase the flash memory. Execute the `load` command to load the sample program. For more information on using these commands, refer to the debug command file "`icdv3_led2.cmd`" included in the sample.
The debug command file "`icdv3_led2.cmd`" includes debug commands that load the flash erase/programming routine, set a vector table address, and write `led2.elf` into flash memory. Specify it in the `-x` option when you start the debugger.
The procedure for starting the debugger from the command prompt is described below.

1. Start the debugger as described above.
2. Set `\EPSON\gnu33\utility\sample_std\dmt33209` to the current directory.
3. Set the path to **gdb.exe**.
4. Execute the command shown below from the command prompt to start the debugger.

```
C:\EPSON\gnu33\utility\sample_std\dmt33209\>gdb -x icdv3_led2.cmd
```

Precautions

When a user program is debugged using the S5U1C33001H and S5U1C33xxxD, keep in mind the following points.

(1) The program to be debugged should be created so that it can be loaded into a free area in the RAM or flash memory of the S5U1C33xxxD before execution. Since the address to which a program is loaded cannot be specified from the debugger, make sure this address is determined when the program is linked. For a memory map in the S5U1C33xxxD, refer to Figure 3.6.1.3.

(2) The entire facility of the S5U1C33001H can be made usable only by connecting it to the PC with a USB cable. Programs are downloaded into RAM is approximately 8 KB/s, and programs are downloaded into flash memory is approximately 7 KB/s. Note, however, that these speeds vary with the PC used and operating conditions.

(3) For other limitations on use of the S5U1C33001H, refer to the document entitled "S1C33 Family In-Circuit Debugger Manual".

3.7 Data Area and Sections

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

3.7.1 Data Area

The area frequently used by a program generally is accessed by first setting its base address in a register and then setting a displacement with the `ext` instruction.
Example:

```
ext     imm13                ; 13-bit displacement (8K bytes)
ld.b    %rd,[%rb]            ; = ld.b  %rd,[%rb+imm13]
```

In this example, an 8K-byte area can be accessed by relative addressing from the base address (`%rb`) using two instructions. If a frequently used variable, etc. is located in this area, the code size of a program can be made smaller and the processing speed can be made faster than possible with other access methods.
Even an area larger than 8K bytes, but not exceeding 64M bytes, can be accessed by a register-indirect addressing with displacement included that uses two `ext` instructions (total three instructions used).
To permit the use of this addressing method, the C/C++ compiler reserves `%r15` for use as the register in which to set the base address (data area pointer). The 64M bytes starting from that address is the default data area.

The following shows the basic method for using the data area.

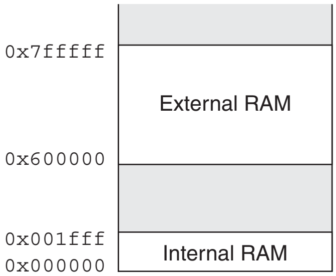


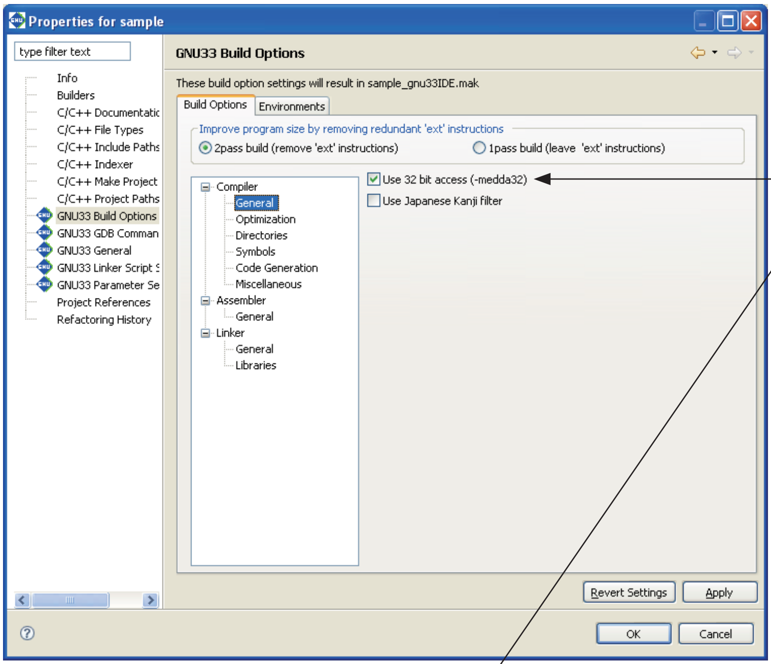
Figure 3.7.1.1 Example of a memory configuration

The internal RAM is assumed to be located in the 64M byte space beginning with the address 0x0, as shown in Figure 3.7.1.1.
Although the discussion addresses RAM only, ROM and I/O memory may also be handled as data areas.
Here, the 64M bytes beginning with the address 0x0 is used as the default data area.

Adding the `-medda32` option of **xgcc** will prevent use of the default data area. This means that the default data area pointer (`__dp`) is not referenced when data is accessed.

1. To use the default data area

In the **IDE**, `-medda32` is specified as a build option of the C/C++ compiler by default. This is the option to permit any location in the entire 32-bit address space to be accessed without using the default data area pointer. If the default data area needs to be used, do not specify this option.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

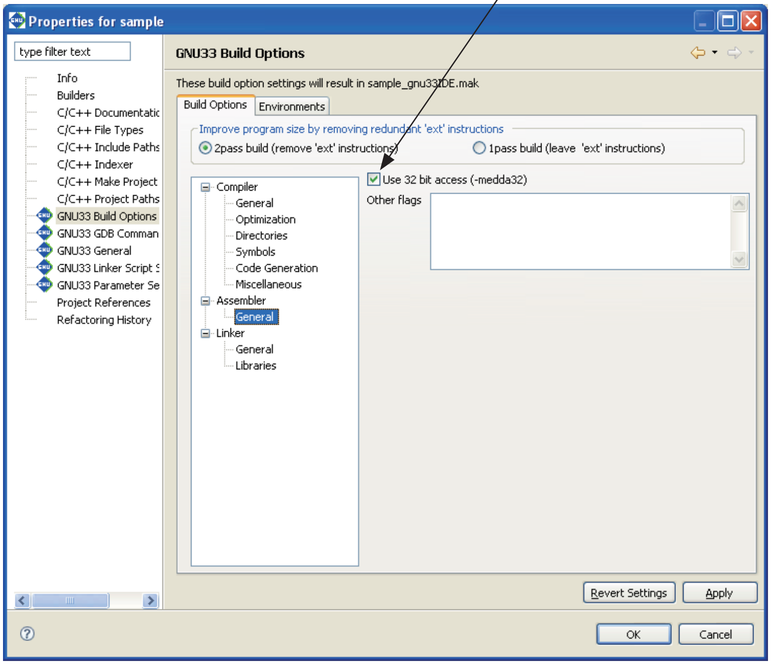
Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

☑ Use 32 bit access (-medda32)

☐ Use Japanese Kanji filter

Deselect the check box


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

☑ Use 32 bit access (-medda32)

Other flags

2. Specifying the start address of the data area

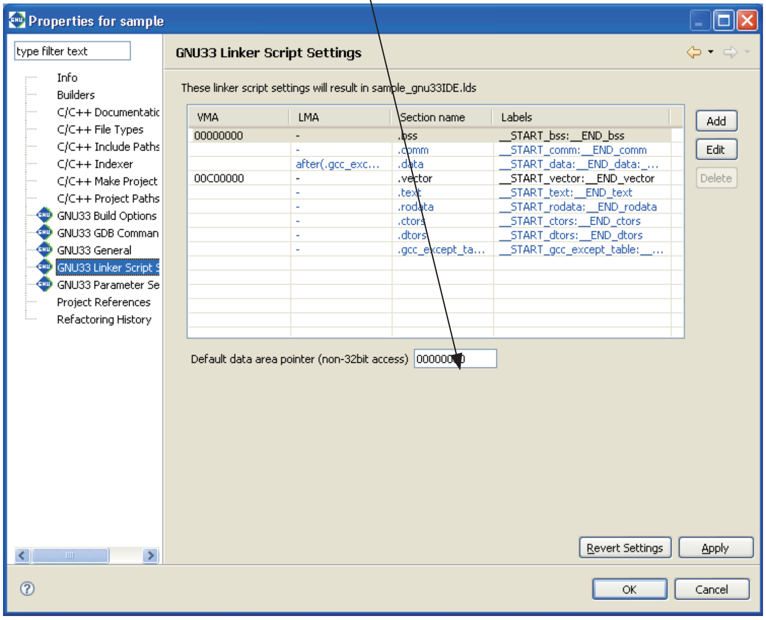
Specifying the data area address in a linker script file

In the linker script file created for use in the linking process, use a data area pointer symbol to specify the start address of the data area.

Example:

```
__dp  = 0x0;
```

In the **IDE**, enter 0 in the [Default data area pointer] text box in [GNU33 Linker Script Settings] of the [Properties] dialog box.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Linker Script Settings
Add
Edit
Delete
Revert Settings
Apply
OK
Cancel

These linker script settings will result in sample_gnu33IDE.lds

VMA	LMA	Section name	Labels
00000000	-	.bss	__START_bss:__END_bss
	-	.comm	__START_comm:__END_comm
	after(.gcc_exc...	.data	__START_data:__END_data:__...
00C00000	-	.vector	__START_vector:__END_vector
	-	.text	__START_text:__END_text
	-	.rodata	__START_rodata:__END_rodata
	-	.ctors	__START_ctors:__END_ctors
	-	.dtors	__START_dtors:__END_dtors
	-	.gcc_except_ta...	__START_gcc_except_table:__...

Default data area pointer (non-32bit access) 00000000

Specifying the data area address in a source file

The register `%r15` used as the default data area pointer is not initialized by the C/C++ compiler. It must be initialized in a user program. You can use the data area pointer symbol defined in a linker script file for this purpose.

Example:

```
xld.w   %r15,__dp           ; Set default data area start address
```

3. External variable definition

All defined external variables are located in the default data area.

4. Result of compilation

A statement referencing an external variable is compiled as shown below.

Example 1:

When **`szDefaultData[0] = 0;`** is compiled without the `-medda32` option

```
xld.w   %r4,0
ext     doff_hi(szDefaultData)
ext     doff_lo(szDefaultData)
ld.b    [%r15],%r4
```

Note that `doff_hi(szDefaultData)` and `doff_lo(szDefaultData)` are the 13 high-order bits and the 13 low-order bits of offset from `__dp` to `szDefaultData`, respectively. Since the default data area is 64M bytes, it is accessed using three instructions.

Example 2:

When **`SzDefaultData[0] = 0;`** is compiled with the `-medda32` option enabled

```
xld.w   %r4,0                ;0x0
xld.w   %r5,SzDefaultData
ld.w    [%r5],%r4
```

As shown above, data is loaded without data pointers.

Example 3:

When **`i = SzDefaultData[0];`** (save instruction) is compiled with the `-medda32` option enabled

```
xld.w   %r4,SzDefaultData
ld.w    %r4,[%r4]
```

As shown above, data is saved without data pointers.

3.7.2 Sections

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.

Eight broad categories (attributes) of sections are set in the **xgcc** C/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.

(5)`.comm` section

Variables without initial values are located here.
The C/C++ compiler determines where to locate variables (`.bss` or `.comm`).
When determining the location of variables without initial values in the assembler, locate them in the `.bss` section.
Only an area in RAM space is reserved for these variables.

(6)`.ctors` section

Pointer arrays to the global class constructor functions are located here.
For C++ sources, these pointer arrays are generated by the C/C++ compiler.
They are eventually written to ROM.

(7)`.dtors` section

Pointer arrays to the global class destructor functions are located here.
For C++ sources, these pointer arrays are generated by the C/C++ compiler.
They are eventually written to ROM.

(8)`.gcc_except_table` section

Table data for exception handling are located here.
When exception handling is used in C++, these table data are generated by the C/C++ compiler.
They are eventually written to ROM.

`.vector` section

The **IDE** has another section with `.rodata` attribute, the `.vector` section, available for use for vector tables.
For C/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 5.7.7, "Editing a Linker Script".

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

Example source files

(file1.s)

```
                :
        .set SP_INI,0x0800   ; sp is in end of 2KB internal RAM

        .text
        .long BOOT           ; BOOT VECTOR
BOOT:
        xld.w %r15,SP_INI
        ld.w  %sp,%r15       ; set SP
        xld.w %r15,__dp      ; set default data area pointer
        xcall main           ; goto main
        jp    BOOT           ;
```

(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)

```
OUTPUT_FORMAT("elf32-c33", "elf32-c33",
              "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);
SECTIONS
{
    __dp = 0x0;                                                          ...(1)

    . = 0x0;                                                             ...(2)
    .bss   : { *(.bss) }                                                 ...(3)
    .data   :
                    AT (__load_data)
        { __start_data = . ; *(.data) ; __end_data = . ; }               ...(4)

    . = 0xc00000;                                                        ...(5)
    .text   : { *(.text) }                                               ...(6)
    .rodata : { *(.rodata) }                                             ...(7)

    __load_data = . ;                                                    ...(8)
}
```

The example source files shown above include the following sections.

`file1`	`.text` section
`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 default data area pointer (`__dp`) to 0x0. This symbol is used to set the `%r15` register in the example source file "`file1.s`".

(2) Set the location counter to 0x0 (start address of RAM). The sections specified thereafter are located from this address.

(3) Define the `.bss` output section to be output to an executable format object file. Note that `{ *(.bss) }` specifies that all `.bss` sections in the input files are to located in the `.bss` section defined here.

(4) Define the `.data` output section. 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 (virtual memory address accessed when actually executing code or when reading/writing data), but this VMA is normally the same as the LMA (load memory address in which data is stored). 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 `__load_data` (ROM location) specified in step (8). Note that `__start_data` and `__end_data` are defined for the start and the end addresses of the `.data` section (VMA) to be referenced later. These symbols are used to copy data from the LMA to the VMA in the example source "`file3.c`".

(5) Set the location counter to 0xc00000 (ROM start address). The sections specified thereafter are located from this address.

(6) Define the `.text` output section for program code. All `.text` sections in the input files are located from the address 0xc00000.

(7) Define the `.rodata` output section. This section is located immediately after the `.text` section. All `.rodata` sections in the input files are located in this output section.

(8) Define the symbol `__load_data` used to specify the LMA of the `.data` section. The current location counter value (address immediately following the `.rodata` section) is defined in this symbol.

Figure 3.7.2.1 shows the memory map configured by this example script.

```
ld -o sample.elf file1.o file2.o file3.o -T sample.lds
```

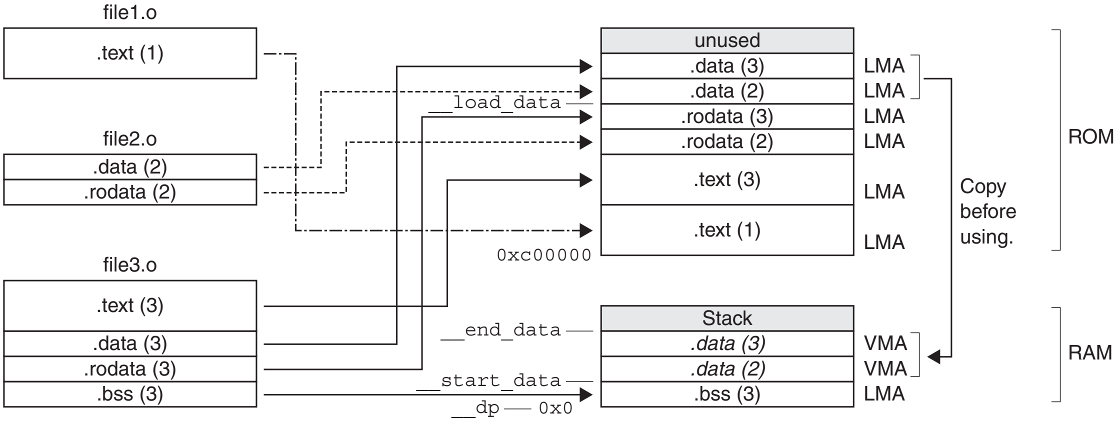

file1.o
.text (1)
file2.o
.data (2)
.rodata (2)
file3.o
.text (3)
.data (3)
.rodata (3)
.bss (3)
__load_data
0xc00000
__end_data
__start_data
__dp
0x0
unused
.data (3)
.data (2)
.rodata (3)
.rodata (2)
.text (3)
.text (1)
Stack
.data (3)
.data (2)
.bss (3)
LMA
LMA
LMA
LMA
LMA
LMA
VMA
VMA
LMA
Copy before using.
ROM
RAM

Figure 3.7.2.1 Memory map configured by `sample.lds`

THIS PAGE IS BLANK.

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 **GNU33 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
C++ source file	<*file name*>.cpp, <*file name*>.cc, <*file name*>.cxx, <*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 S1C33 tools.
Make sure the extension of the C source file is ".c" (small letter can only be used).

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.

File size

The following shows the guide about the upper limit of the C/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.)

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/C++ Source

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

4.2.1 Data Type

The **xgcc** C/C++ compiler supports all data types under ANSI C/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	4	-2147483648 to 2147483647
unsigned int	4	0 to 4294967295
long	4	-2147483648 to 2147483647
unsigned long	4	0 to 4294967295
pointer	4	0 to 4294967295
float	4	1.175e-38 to 3.403e+38 (normalized number)
double	8	2.225e-308 to 1.798e+308 (normalized number)
long long	8	-9223372036854775808 to 9223372036854775807
unsigned long long	8	0 to 18446744073709551615
bool	1	true/false
wchar_t (C)	2	0 to 65535
wchar_t (C++)	4	-2147483648 to 2147483647

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, a warning 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;
     →warning: integer constant is too large for "long" type
ll_val = 0x1234567812345678LL;
     →OK
```

The type `bool` is the data type needed to handle determination of `true` or `false`. C requires the inclusion of `stdbool.h`. In C++, `bool` is an independent type.

Type `wchar_t` is the data type needed to handle wide characters. In C, this data type is defined in `stdlib.h`/`stddef.h` as the type `unsigned short`.

In C++, this data type is a signed four-byte independent type. To use `wchar_t` in C++, specify the prefix `L`. This specification is required to have the compiler recognize wide character strings as such.

Example: `wchar_t buf[] = L"123";`

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*, 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, vprintf, 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
	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
	assert	assert.h
	setlocale*	locale.h
libgcc2.a	strtoll, strtoull	stdlib.h

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

Emulation library

File name	Functions
libgcc.a	adddf3, subdf3, muldf3, divds3, negdf2, addds3, subds3, mulds3, divds3, negds2, fixunsdfsi, fixdfsi, floatsidf, fixunssfsi, fixsfsi, floatsisf, truncdfsf2, extendsfdf2, fcmpd, fcmps, divsi3, udivsi3, modsi3, umodsi3

`long long` **type emulation library**

File name	Functions
libgcc2.a	muldi3, divdi3, udivdi3, moddi3, umoddi3, negdi2, lshrdi3, ashrdi3, ashldi3, fixunsdfdi, fixdfdi, floatdidf, fixunssfdi, fixsfdi, floatdisf, cmpdi2, ucmpdi2, ffsdi2

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/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\
      .space    8\n\
      .long     ZERO_DIV\n\
      .space    4\n\
      .long     ADDR_ERR\n\
      .long     NMI\n\
      .space    32\n\
      .long     INT1\n\
      .long     INT2");
```

Example 3:
```
BOOT(){
      asm("xld.w %r15,0x0800");
      asm("ld.w  %sp,%r15");     /* set SP */
      asm("ld.w  %r15,0x0000"); /* set data area pointer */
            :
}
```

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:
      pushn   %r2
      xld.w   %r2,5        ;0x5
      xld.w   %r1,int_num
      ld.w    [%r1],%r2
      popn    %r2
      reti
```

4.2.5 `alloca()`/Variable Length Array

The C/C++ compiler **xgcc** supports `alloca()`/variable length arrays.
To use `alloca()`, be sure to include `alloca.h`. Since `alloca()` does not check for stack overflows, monitor stack size at this point in time.
When `alloca()`/variable length array is used, a frame pointer is assigned to `%r0`.
Example: `#include <alloca.h>`

```
int main()
{
   char* cp_pt;

   cp_pt = (char*)alloca(3);
   cp_pt[0] = 5;
   cp_pt[1] = 6;
   cp_pt[2] = 7;

   return 0;
   }
```

Assembler code

```
main:
   pushn    %r0
   ld.w     %r0,%sp      ← The value of %sp is saved in the frame pointer (%r0).
   ld.w     %r5,%sp
   ld.w     %r4,%sp
   xsub     %r4,4        ;0xfffffffc
   ld.w     %sp,%r4
   xld.w    %r4,5        ;0x5
   ld.b     [%sp],%r4
   xld.w    %r4,6        ;0x6
   xld.b    [%sp+1],%r4
   xld.w    %r4,7        ;0x7
   xld.b    [%sp+2],%r4
   ld.w     %sp,%r5
   xld.w    %r4,0        ;0x0
   popn     %r0
   ret
```

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   2002.2.14                                               4
; boot program                                                     4

#define    SP_INI,0x0800           ; Stack pointer value           6

           .text                                                   2
           .long  BOOT             ; BOOT VECTOR                   2
BOOT:                                                              3
           xld.w  %r15,SP_INI                                      5
           ld.w   %sp,%r15         ; set SP                        1
           ld.w   %r15,__dp        ; set data area pointer         1
           xcall  main             ; goto main                     5
           xjp    BOOT             ; infinity loop                 5
  :             :                        :
```

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.w %r1,%r2
           ld.w %r0,%r1
  ;Error
  BOOT:    ld.w %r1,%r2        ld.w   %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.w %r1,%r2
  ;Error
           ld.w %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 C33 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.w  %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 core instructions can be used in the S1C33 Family:

C33 STD Core

```
nop       slp       halt      pushn     popn      brk       retd
int       reti      call      call.d    ret       ret.d     jp
jp.d      jrgt      jrgt.d    jrge      jrge.d    jrlt      jrlt.d
jrle      jrle.d    jrugt     jrugt.d   jruge     jruge.d   jrult
jrult.d   jrule     jrule.d   jreq      jreq.d    jrne      jrne.d
ld.b      ld.ub     ld.h      ld.uh     ld.w      add       sub
cmp       and       or        xor       not       srl       sll
sra       sla       rr        rl        scan0     scan1     swap
mirror    div0s     div0u     div1      div2s     div3s     btst
bclr      bset      bnot      adc       sbc       mlt.h     mltu.h
mlt.w     mltu.w    mac       ext
```

C33 ADV Core

```
jpr       jpr.d     pushs     pops      mac.w     mac.hw    macclr
ld.cf     div.w     divs.w    repeat    retm      swaph     sat.b
sat.ub    mlt.hw    mac1.h    mac1.hw   mac1.w    ld.c      sat.h
sat.uh    sat.w     sat.uw    loop      do.c      psrset    psrclr
ext
```

C33 PE Core

```
push      pop       pushs     pops      jpr       jpr.d     psrclr
psrset    swaph     ld.c      do.c      ld.cf
```

Refer to the C33 Core manuals 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.w`", "`LD.W`", and "`Ld.w`" are all accepted as "`ld.w`" 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.

`.text`		Declares a `.text` section.
`.section .data`		Declares a `.data` section.
`.section .rodata`		Declares a `.rodata` section.
`.section .bss`		Declares a `.bss` section.
`.long`	<*data*>	Defines a 4-byte data.
`.short`	<*data*>	Defines a 2-byte data.
`.byte`	<*data*>	Defines a byte data.
`.ascii`	<*string*>	Defines an ASCII character strings.
`.space`	<*length*>	Defines a blank (0x0) space.
`.zero`	<*length*>	Defines a blank (0x0) space.
`.align`	<*value*>	Alignment to a specified boundary address.
`.global`	<*symbol*>	Defines a global symbol.
`.set`	<*symbol*>**,**<*address*>	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:          1label:
_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.w %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 C33 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–R15	`%r0–%r15` or `%R0–%R15`

Special register (%sd, %ss)	Notation
Processor status register PSR	`%psr` or `%PSR`
Stack pointer SP	`%sp` or `%SP`
Arithmetic operation low register ALR	`%alr` or `%ALR`
Arithmetic operation high register AHR	`%ahr` or `%AHR`

Special register (%sd) in C33 ADV Core	Notation
Loop counter	`%lco` or `%LCO`
Loop-start-address register	`%lsa` or `%LSA`
Loop-end-address register	`%lea` or `%LEA`
Shift-out register	`%sor` or `%SOR`
Vector table base register	`%ttbr` or `%TTBR`
Data pointer	`%dp` or `%DP`
Processor identification register	`%idir` or `%IDIR`
Debug base register	`%dbbr` or `%DBBR`
User stack pointer	`%usp` or `%USP`
Supervisor stack pointer	`%ssp` or `%SSP`

Special register (%sd) in C33 PE Core	Notation
Vector table base register	`%ttbr` or `%TTBR`
Processor identification register	`%idir` or `%IDIR`

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

Examples: `[%r8]`   `[%r1]+`   `[%sp+imm6]`

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
imm2	2-bit immediate data	0 to 3	0x0 to 0x3	0b0 to 0b11
imm3	3-bit immediate data	0 to 7	0x0 to 0x7	0b0 to 0b111
imm4	4-bit immediate data	0 to 15	0x0 to 0xf	0b0 to 0b1111
imm6	6-bit immediate data	0 to 63	0x0 to 0x3f	0b0 to 0b111111
sign6	Signed 6-bit immediate data	-32 to 31	0x0 to 0x3f	0b0 to 0b111111
imm8	8-bit immediate data	0 to 255	0x0 to 0xff	0b0 to 0b11111111
sign8	Signed 8-bit immediate data	-128 to 127	0x0 to 0xff	0b0 to 0b11111111
imm10	10-bit immediate data	0 to 1023	0x0 to 0x3ff	0b0 to 0b1111111111
imm13	13-bit immediate data	0 to 8191	0x0 to 0x1fff	0b0 to 0b1111111111111
imm32	32-bit immediate data	0 to 4294967295	0x0 to 0xffffffff	0b0 to 0b11111111111111111111111111111111
sign32	Signed 32-bit immediate data	-2147483648 to 2147483647	0x0 to 0xffffffff	0b0 to 0b11111111111111111111111111111111

(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 32-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 0x0000ff00.

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 = sign8
ld.w    %rd,LABEL   ← LABEL = sign6
```

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

LABEL + imm32 LABEL + sign32

Example:
```
call    LABEL+0x10
```

Symbol mask

The basic instructions in the C33 Core instruction set are characterized by the fact that the immediate size that can be specified in the operand of each instruction is limited. Consequently, an assembler error results when a symbol whose value exceeds the size is used. When using the basic instructions, the high-order bits must be written separately in the `ext` instruction. A symbol mask is used for this purpose.

Specifically, a symbol mask is used to get the values from a symbol value that are written separately in the `ext` instruction and the basic instruction, and is entered immediately after the symbol.

When using extended instructions, the **as** assembler attaches the necessary symbol mask as it expands the instruction. Therefore, you do not specifically need to be concerned about the `ext` instruction or symbol mask.

Types of symbol masks

The following 8 types of symbol masks can be used:

Symbol mask	Function
`@rh` or `@RH`	Acquires the 10 high-order bits of a relative address.
`@rm` or `@RM`	Acquires the 13 mid-order bits of a relative address.
`@rl` or `@RL`	Acquires the 8 low-order bits of a relative address.
`@h` or `@H`	Acquires the 13 high-order bits of an absolute address.
`@m` or `@M`	Acquires the 13 mid-order bits of an absolute address.
`@l` or `@L`	Acquires the 6 low-order bits of an absolute address.
`@ah` or `@AH`	Acquires the 13 high-order bits of a relative address.
`@al` or `@AL`	Acquires the 13 low-order bits of a relative address.

Examples:

```
ext   LABEL@rh
ext   LABEL@rm
call  LABEL@rl          ← Functions as "call LABEL".

ext   LABEL@h
ext   LABEL@m
ld.w  %rd,LABEL@l       ← Functions as "ld.w %rd,LABEL".

ext   LABEL@ah
ext   LABEL@al
ld.w  %rd,[%rb]         ← Functions as "ld.w %rd,[%rb+LABEL]".
```

Notes:
- The symbol masks are effective only on the defined symbols. If a symbol mask is applied to a numeric value, an error will result.
- If a symbol mask is omitted, the lower bits effective for that instruction will be used. However, if the bit value does not fall within the instruction range, an error or warning will be issued.

Pseudo-operand

To specify offset addresses from the default data area start address, the **as** assembler provides the pseudo-operands that can be used as symbol masks.

Types of pseudo-operands

The following 5 types of pseudo-operands can be used:

Pseudo-operand	Function
`doff_hi()` or `DOFF_HI()`	Acquires the 13 high-order bits of an offset from the default data area start address (`%r15`; `__dp`).
`doff_lo()` or `DOFF_LO()`	Acquires the 13 low-order bits of an offset from the default data area start address (`%r15`; `__dp`).
`dpoff_h()` or `DPOFF_H()`	Acquires the 13 high-order bits of an offset from the default data area start address (`%dp`; `__dp`). This is for the C33 ADV Core and can be used when the `-mc33adv` option is specified.
`dpoff_m()` or `DPOFF_M()`	Acquires the 13 mid-order bits of an offset from the default data area start address (`%dp`; `__dp`). This is for the C33 ADV Core and can be used when the `-mc33adv` option is specified.
`dpoff_l()` or `DPOFF_L()`	Acquires the 6 low-order bits of an offset from the default data area start address (`%dp`; `__dp`). This is for the C33 ADV Core and can be used when the `-mc33adv` option is specified.

Examples:

```
ext   doff_hi(FOO)
ext   doff_lo(FOO)
ld.w  %r0,[%r15]        ← Functions as "ld.w %r0,[%r15+(FOO address-__dp)]".
```

Notes:
- The pseudo-operands are effective only on the defined symbols. If a pseudo-operand is applied to a numeric value, an error will result.
- When using pseudo-operands, the default data area start address must be set to the corresponding default data area pointers (see Section 3.7.1, "Data Area").

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

C33 STD Core instructions

```
xadd      xsub      xcmp      xand      xoor      xxor      xnot
xsll      xsrl      xsla      xsra      xrl       xrr       xld.b
xld.ub    xld.h     xld.uh    xld.w     xbset     xbclr     xbtst
xbnot     xjp       xjreq     xjrne     xjrgt     xjrge     xjrlt
xjrle     xjrugt    xjruge    xjrult    xjrule    xcall     xjp.d
xjreq.d   xjrne.d   xjrgt.d   xjrge.d   xjrlt.d   xjrle.d   xjrugt.d
xjruge.d  xjrult.d  xjrule.d  xcall.d
sjp       sjreq     sjrne     sjrgt     sjrge     sjrlt     sjrle
sjrugt    sjruge    sjrult    sjrule    scall     sjp.d     sjreq.d
sjrne.d   sjrgt.d   sjrge.d   sjrlt.d   sjrle.d   sjrugt.d  sjruge.d
sjrult.d  sjrule.d  scall.d
```

C33 ADV Core instructions

```
ald.b     ald.ub    ald.h     ald.w
```

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@rh
                        ; ext  LABEL@rm
                        ; call LABEL@rl

xld.w %r1,sign32        ; ext  sign32@h
                        ; ext  sign32@m
                        ; ld.w %r1,sign32@l
```

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.w %r0,[symbol + 0x10]  ; R0 ← [symbol + 0x10]
xjp   LABEL + 5            ; Jumps to address LABEL + 5.
xrl   %r0,15               ; Rotates the R0 content left by 15 bits.
```

For details about the extended instructions that include operands, refer to Section 8.7, "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:

`#include`	Insertion of file
`#define`	Definition of character strings and numbers
`#if-#else-#endif`	Conditional assembly

Examples:

```
#include   "define.h"
#define    NULL     0
#ifdef     TYPE1
           ld.w     %r0,0
#else
           ld.w     %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/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/C++ and assembler modules in a program, pay attention to the interface between the C/C++ functions and assembler routines, such as arguments, size of return values and the parameter passing conventions.

(4) If the target uses the C33 ADV Core, always confirm that the OC bit of the `%psr` register is set to 1. (The OC bit is initially set to 0.) Executing an arithmetic/logic instruction when OC bit = 1 will clear the V bit of `%psr` to 0. The C/C++ compiler is created based on this bit operation.
Example settings during boot:

```
asm("xld.w %r15,0x2000");          // Set SP in end of 8KB internal RAM
asm("ld.w  %sp,%r15");
asm("xld.w %r15,0x00200000");      // OC=1 // Essential
asm("ld.w  %psr,%r15");            // Essential
asm("xld.w %r15,__dp");            // Set DP (default data area)
asm("ld.w  %dp,%r15");
```

(5) Be sure to include the prototype declaration or the extern declaration of the functions.
If there is no prototype or extern declaration and if a function without its definition part in an earlier part of the same file is called, the type assumed in the file calling the function may differ from the function type actually called, resulting in a potential malfunction. Even so, the function will compile without errors.
However, a warning is generated if the definition part of the called function is present in the same file. If the definition part of the called function is located in another file, no warning is generated unless the -Wall option is attached.
Since the return value is implicitly assumed to be of the int type, the correct value will not be returned if the return value has a data type larger than int.
Example:

```
long long ll_Val_1 = 0x1122334455667788LL;
long long ll_Val_2;

int main()
  {
        ll_Val_2 = sub();     // ll_Val_2 is substituted with 0x55667788.
        return 0;
  }

long long sub()
  {
        long long ll_wk;

        ll_wk = ll_Val_1;
        return ll_wk;
  }
```

(6) Do not use a pointer other than char type for reading/writing an odd memory address, as it will cause an address misaligned exception be occurred.
Example:

```
int *ip_Pt;

void sub()
{
  ip_Pt = (int *)0x3;
  (*ip_Pt) = 0x2;              // An address misaligned exception occurs.
}
```

(7) In the C/C++ source, a function name can be used as a pointer to the function, but a function name cannot be used to assign its value to a real-type (float/double) variable or array.

A function name can be used to assign a value to an integer-type variable or array.

In the C source, however, when a function name is used to assign a value at the time of declaration of a global variable or global array, it can be used to assign a value only to a global variable or array of the int/unsigned int/long/unsigned long type.

At a time other than declaration, a function name can be used to assign a value to any integer-type global variable or array.

In the case of local variables or local arrays, as long as they are of the integer type, a function name can be used to assign a value at the time of declaration or at any other time.

Example:

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

→ A function name can be used to assign a value at the time of declaration of long-type global variable, `l_Global_Val`.

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

→ An error is generated if function name is used to assign a value at the time of declaration of short-type global variable, `s_Global_Val`.

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

In the C++ source, the following warning is displayed instead of an error message.

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

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

→ A function name can be used to assign a value to long-type local variable, `l_local_val`.

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

→ A function name can be used to assign a value to short-type local variable, `s_local_val`.

5)
```
char c_Global_Val;
void sub()
{
   c_Global_Val = (char)boot;
}
```

→ At a time other than declaration, a function name can be used to assign a value to char-type global variable, c_Global_Val.

(8) In the C/C++ source, a function pointer can also be used, but it cannot be used to assign a value to a real-type (float/double) variable or array as described in (7).

A function pointer can be used to assign a value to an integer type variable or array.

In the C source, however, a function pointer cannot be used to assign a value at the time of declaration of a global variable or global array.

At a time other than declaration, a function pointer can be used to assign a value to an integer-type global variable or array.

In the case of integer-type local variables and local arrays, a function pointer can be used to assign a value at the time of declaration or at any other time.

A warning is generated if a function pointer is used to assign a value to a variable or array that is not an int/unsigned int/long/unsigned long type.

Example:

void (* fp_Pt)(void); // Declaration of function pointer for which both return value and argument are void type

1) `long l_Global_Val = (long)fp_Pt;`

→ An error is generated if function pointer is used to assign a value at the time of declaration of global variable, `l_Global_Val`.

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

(An error is not generated in C++ source)

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

→ A function pointer can be used to assign a value to long-type local variable, `l_local_val`.

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

→ A function pointer can be used to assign a value to short-type local variable, `s_local_val`, but a warning is generated.

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

4)
```
long l_Global_Val;
void sub()
{
   l_Global_Val = (long)fp_Pt;
}
```

→ At a time other than declaration, a function pointer can be used to assign a value to long-type global variable, `s_Global_Val`.

(9) Note that, because of the specifications of the C/C++ language, if a process with an undefined operation is executed, the result may vary depending on the optimization option (`-O0/-O/-O2/-O3/-Os`) and differences in local variables/external variables.

Undefined processes include the following:

- Overflow resulting from conversion of floating point type value to integer type value
- Shift operation conducted with a negative value or with a bit length that is the same as or longer than the bit length used in calculation after a type promotion

(10)Because of the specifications of the C language, accessing a variable by means of a non-interchangeable pointer may display the following warning message if the -Wall option is attached.

In this case, reference or assignment of variables via pointers may not be performed correctly.

```
warning: dereferencing type-punned pointer will break strict-aliasing
rules
```

Example: When -O3 option is attached

```
int sub()
{
      int p1 = 0 ;
      short *p2 = (short *)&p1 ;
```

Since p2 and &p1 are non-interchangeable, a warning will be generated.

In that case, variable p1 may not be referenced by means of pointer p2 or assignment may not be performed correctly.

5 GNU33 IDE

This chapter describes the facilities available with the **GNU33 IDE** and describes how to use the **GNU33 IDE**.

5.1 Overview

5.1.1 Features

The **GNU33 IDE** (hereafter simply the **IDE**) provides an integrated development environment that makes it user to develop software using the S1C33 Family C/C++ Compiler Package (S5U1C33001C).
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, C++, and assembler
  The **IDE** lets users create and edit sources in GNU-compliant C, 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	3.4.0
Eclipse CDT	5.0.0

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 GNU33 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.)

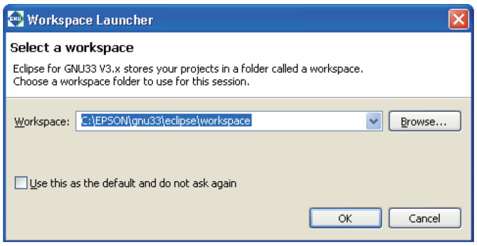
5.2 Starting and Quitting the IDE

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\gnu33\eclipse directory to start the **IDE**.
You can also start the **IDE** by selecting [EPSON MCU] > [GNU33] > [GNU33 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.



Workspace Launcher

Select a workspace

Eclipse for GNU33 V3.x stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\EPSON\gnu33\eclipse\workspace

☐ Use this as the default and do not ask again

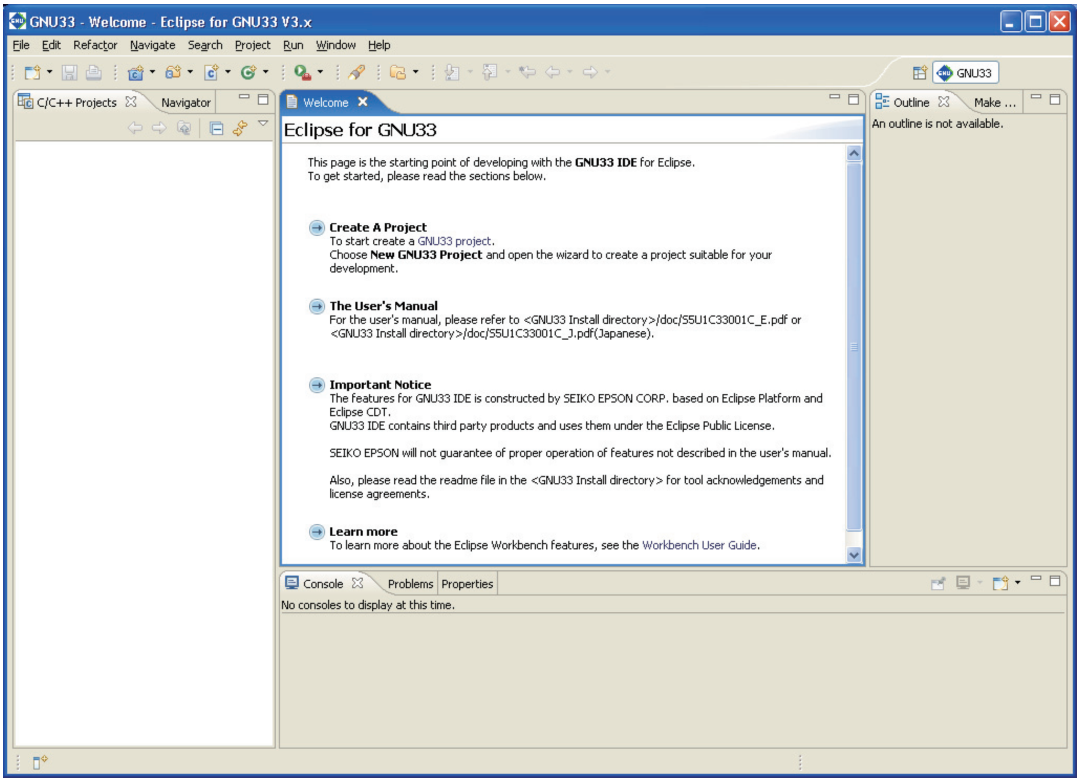
Although c:\EPSON\gnu33\eclipse\workspace is displayed as the default directory, you can select any other directory or create a new directory and set it as the workspace. Enter a directory name in the [Workspace:] combo box or select one from the directory select dialog box displayed by clicking the [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.)

* Do not specify the project directory (directory containing .project file) as a workspace directory. Doing so may result in failures with project imports (when [Copy projects into workspace] is selected).
The current workspace directory can be checked by selecting [File] > [Switch workspace...] > [Others...] and opening the [Workspace Launcher] dialog box.

(3) Click the [OK] button.

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


GNU33 - Welcome - Eclipse for GNU33 V3.x
File Edit Refactor Navigate Search Project Run Window Help
GNU33
C/C++ Projects
Navigator
Welcome
Outline
Make ...
Console
Problems
Properties

Eclipse for GNU33

This page is the starting point of developing with the **GNU33 IDE** for Eclipse.
To get started, please read the sections below.

Create A Project
To start create a GNU33 project.
Choose **New GNU33 Project** and open the wizard to create a project suitable for your development.

The User's Manual
For the user's manual, please refer to <GNU33 Install directory>/doc/S5U1C33001C_E.pdf or <GNU33 Install directory>/doc/S5U1C33001C_J.pdf(Japanese).

Important Notice
The features for GNU33 IDE is constructed by SEIKO EPSON CORP. based on Eclipse Platform and Eclipse CDT.
GNU33 IDE contains third party products and uses them under the Eclipse Public License.

SEIKO EPSON will not guarantee of proper operation of features not described in the user's manual.

Also, please read the readme file in the <GNU33 Install directory> for tool acknowledgements and license agreements.

Learn more
To learn more about the Eclipse Workbench features, see the Workbench User Guide.

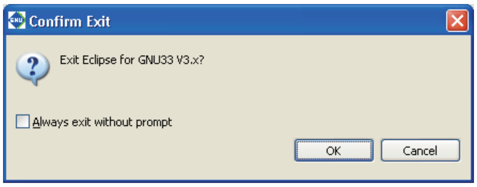
An outline is not available.

No consoles to display at this time.

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.


Confirm Exit
OK
Cancel

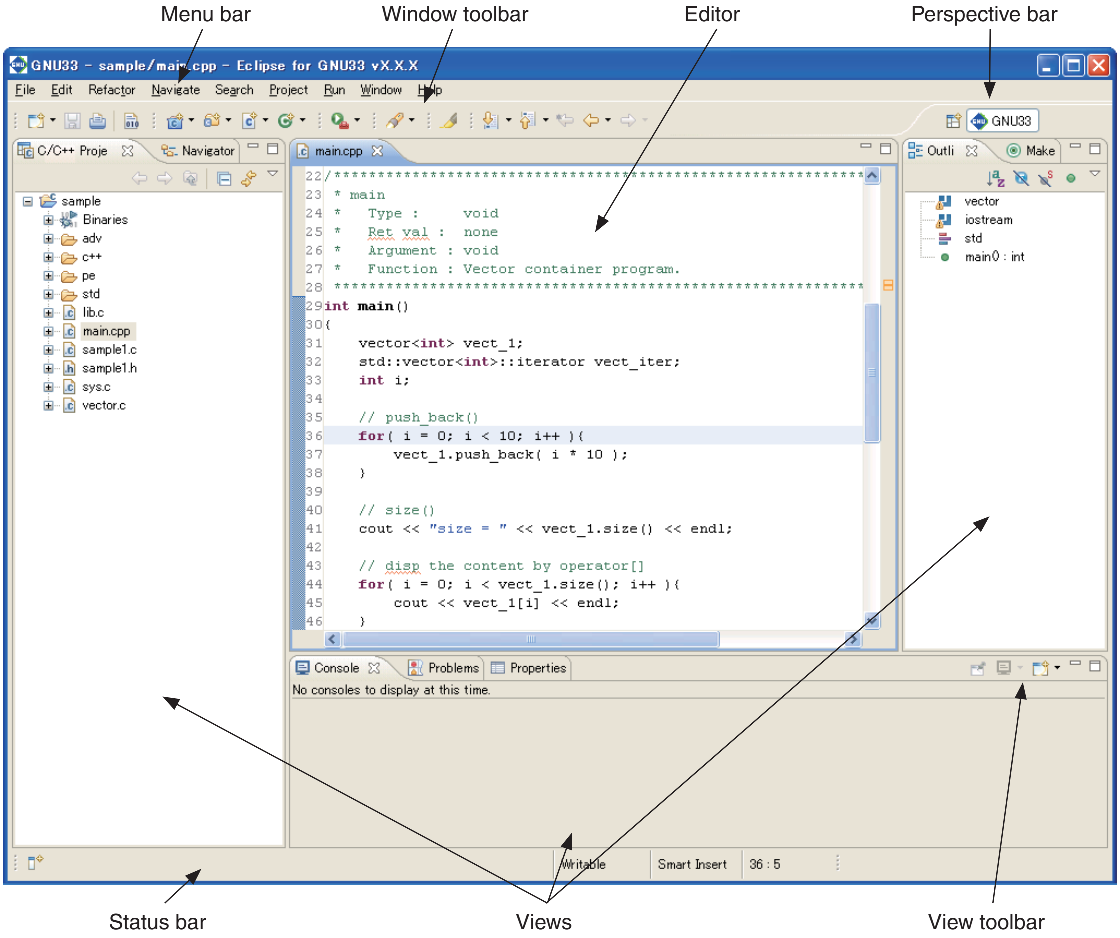
Exit Eclipse for GNU33 V3.x?

☐ Always exit without prompt

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.

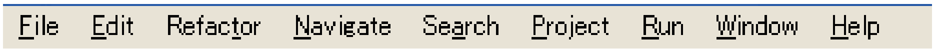

Menu bar
Window toolbar
Editor
Perspective bar
GNU33 - sample/main.cpp - Eclipse for GNU33 vX.X.X
File
Edit
Refactor
Navigate
Search
Project
Run
Window
Help
GNU33
C/C++ Proje
Navigator
sample
Binaries
adv
c++
pe
std
lib.c
main.cpp
sample1.c
sample1.h
sys.c
vector.c
main.cpp
Outli
Make
vector
iostream
std
main() : int
Console
Problems
Properties
Status bar
Views
View toolbar

```
/*********************************************************************
 * main
 *   Type :     void
 *   Ret val :  none
 *   Argument : void
 *   Function : Vector container program.
 *********************************************************************
int main()
{
    vector<int> vect_1;
    std::vector<int>::iterator vect_iter;
    int i;

    // push_back()
    for( i = 0; i < 10; i++ ){
        vect_1.push_back( i * 10 );
    }

    // size()
    cout << "size = " << vect_1.size() << endl;

    // disp the content by operator[]
    for( i = 0; i < vect_1.size(); i++ ){
        cout << vect_1[i] << endl;
    }
```

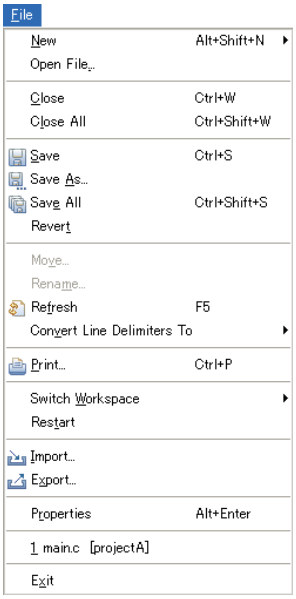
No consoles to display at this time.

Writable | Smart Insert | 36 : 5

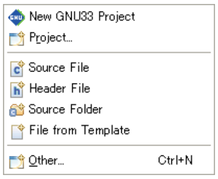
5.3.1 Menu Bar


File
Edit
Refactor
Navigate
Search
Project
Run
Window
Help

[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...
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
1 main.c [projectA]
Exit

New (Alt+Shift+N)


New GNU33 Project
Project...
Source File
Header File
Source Folder
File from Template
Other... Ctrl+N

New GNU33 Project

Launches a GNU33 wizard to create a new project.

Project...

Selects a wizard to create a new project. Select [GNU33 Project] > [New GNU33 Project] from the various wizards shown in the displayed dialog box.

Source File

Creates a new source file.

Header File

Creates a new header file.

Source Folder

Creates a new source folder.

File from Template

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 opens. The [Save All] processing is executed for the file currently opened by the editor.

* Do not specify the project directory (directory containing .project file) as a workspace directory.

Restart

Restarts the IDE.

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.

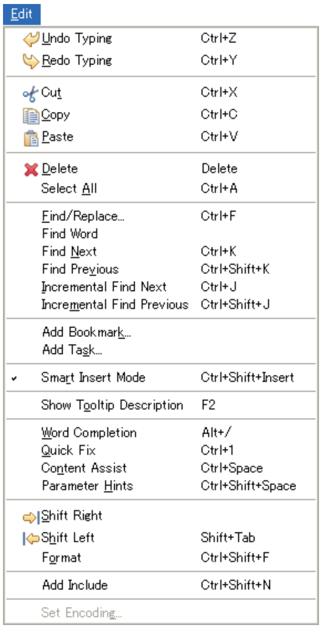
Properties (Alt+Enter)

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
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Select All Ctrl+A
Find/Replace... Ctrl+F
Find Word
Find Next Ctrl+K
Find Previous Ctrl+Shift+K
Incremental Find Next Ctrl+J
Incremental Find Previous Ctrl+Shift+J
Add Bookmark...
Add Task...
✓ Smart Insert Mode Ctrl+Shift+Insert
Show Tooltip Description F2
Word Completion Alt+/
Quick Fix Ctrl+1
Content Assist Ctrl+Space
Parameter Hints Ctrl+Shift+Space
Shift Right
Shift Left Shift+Tab
Format Ctrl+Shift+F
Add Include Ctrl+Shift+N
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 word

Searches for the next occurrence that matches the search string.

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 [↑] or [↓]. You can cancel this search mode by pressing the arrow keys [←] or [→] 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.

Smart Insert Mode (Ctrl+Shift+Insert)

Changes the editor Smart Insert Mode.

Show Tooltip Description (F2)

Pressing the [F2] key while a tooltip is displayed focuses on the tooltip.

Word Completion (Alt+/)

Inserts a word beginning with the character being entered in the editor into the current position. The most recently entered word is selected.

Quick Fix (Ctrl+1)

Displays proposals for error/warning corrections.

Content Assist (Ctrl+Space)

Displays a dialog box and inserts the C source keyword or template selected in the dialog box into the current cursor position in the editor. This is possible only during the editing of a C source.

Parameter Hints (Ctrl+Shift+Space)

Displays a tip for function arguments.

Shift Right

Moves the beginning of a line by one tab to the right.

Shift Left (Shift+Tab)

Moves the beginning of a line by one tab to the left.

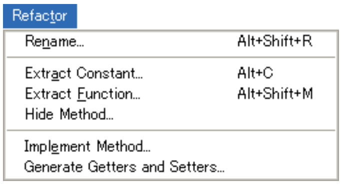
Format (Ctrl+Shift+F)

Formats a text according to formatter settings.

Set Encoding...

Selects the text-encoding format.

[Refactor] menu


Refactor
Rename... Alt+Shift+R
Extract Constant... Alt+C
Extract Function... Alt+Shift+M
Hide Method...
Implement Method...
Generate Getters and Setters...

Rename... (Alt+Shift+R)

Changes the name of a selected function or variable.

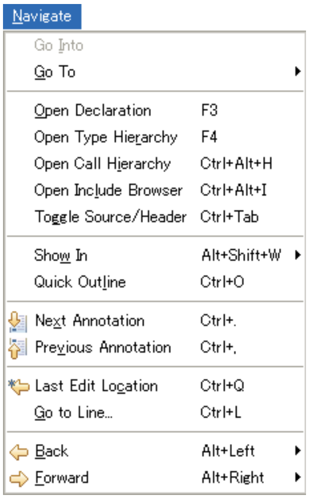
Extract Constant (Alt+C)

Cuts out a constant from a source file for use in a variable.

Extract Function (Alt+Shift+M)

Cuts out a part of a code from a source file for use in a function.

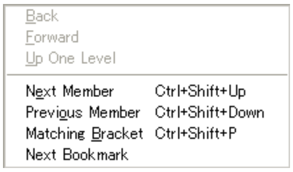
[Navigate] menu


Navigate
Go Into
Go To
Open Declaration F3
Open Type Hierarchy F4
Open Call Hierarchy Ctrl+Alt+H
Open Include Browser Ctrl+Alt+I
Toggle Source/Header Ctrl+Tab
Show In Alt+Shift+W
Quick Outline Ctrl+O
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
Next Member Ctrl+Shift+Up
Previous Member Ctrl+Shift+Down
Matching Bracket Ctrl+Shift+P
Next Bookmark

Back

Returns the [C/C++ Projects] or [Navigator] view to the one displayed immediately before.

Forward

Returns the display traced back by [Go To] > [Back] above to the next most recent state.

Up One Level

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

Next Member (Ctrl+Shift+Up)

Jumps to the next function or variable defining location. (C editor only)

Previous Member (Ctrl+Shift+Dowa)

Jumps to the previous function or variable defining location. (C editor only)

Matching Bracket (Ctrl+Shift+P)

Jumps to a matching bracket. (C editor only)

Next Bookmark

Jumps to the next bookmark. (C editor only)

Open Declaration (F3)

Opens the declaration or definition of a selected object. (Effective when the indexer is ON)

Open Type Hierarchy (F4)

Opens the type hierarchy of a selected variable. (Effective when the indexer is ON)

Open Call Hierarchy (Ctrl+Alt+H)

Opens the call hierarchy of a selected function. (Effective when the indexer is ON)

Open Include Browser (Ctrl+Alt+I)

Opens the include hierarchy of a selected source file. (Effective when the indexer is ON)

Toggle Source/Header (Ctrl+Tab)

Switches to the corresponding source file and header file with the editor.

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.

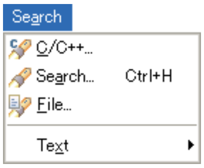
Back (Alt+Left)

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
C/C++...
Search... Ctrl+H
File...
Text

Search... (Ctrl+H)

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

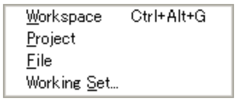
File...

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

C/C++...

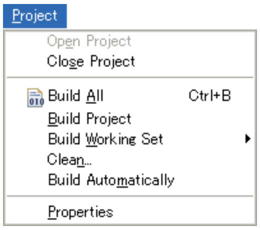
Searches for C source containing the specified string. (Displays the C search page of the [Search] dialog box.)

Text


Workspace Ctrl+Alt+G
Project
File
Working Set...

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


Project
Open Project
Close Project
Build All Ctrl+B
Build Project
Build Working Set
Clean...
Build Automatically
Properties

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.

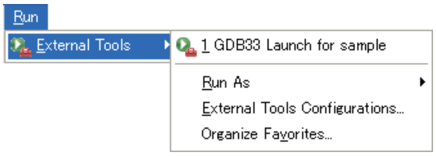
Build Automatically

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


Run
External Tools
1 GDB33 Launch for sample
Run As
External Tools Configurations...
Organize Favorites...

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.

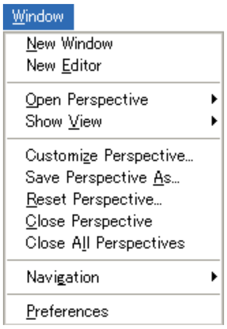
External Tools Configurations...

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


Window
New Window
New Editor
Open Perspective
Show View
Customize Perspective...
Save Perspective As...
Reset Perspective...
Close Perspective
Close All Perspectives
Navigation
Preferences

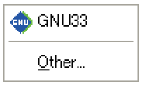
New Window

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


GNU33
Other...

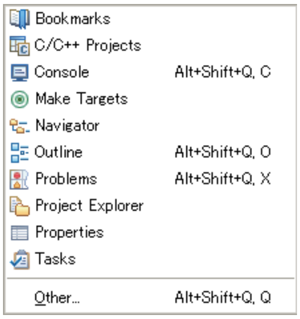
GNU33

Opens the GNU33 perspective.

Other...

Opens another perspective.

Show View


Bookmarks
C/C++ Projects
Console Alt+Shift+Q, C
Make Targets
Navigator
Outline Alt+Shift+Q, O
Problems Alt+Shift+Q, X
Project Explorer
Properties
Tasks
Other... Alt+Shift+Q, Q

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.

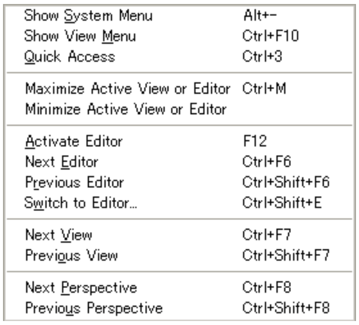
Close Perspective

Closes the currently active perspective.

Close All Perspectives

Closes all loaded perspectives.

Navigation


Show System Menu Alt+-
Show View Menu Ctrl+F10
Quick Access Ctrl+3
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
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).

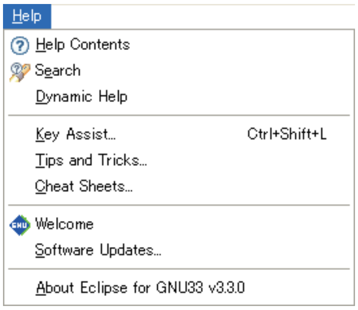
Previous Perspective (Ctrl+Shift+F8)

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

Preferences...

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

[Help] menu


Help
Help Contents
Search
Dynamic Help
Key Assist... Ctrl+Shift+L
Tips and Tricks...
Cheat Sheets...
Welcome
Software Updates...
About Eclipse for GNU33 v3.3.0

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.

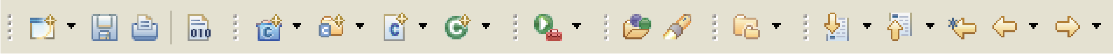
Software Updates

Installs an updater, updates, plug-ins, etc. for software management.
Use this command only when required.

About Eclipse for GNU33 V*x.x*

Shows **IDE** version information and detailed information on plug-ins, etc.

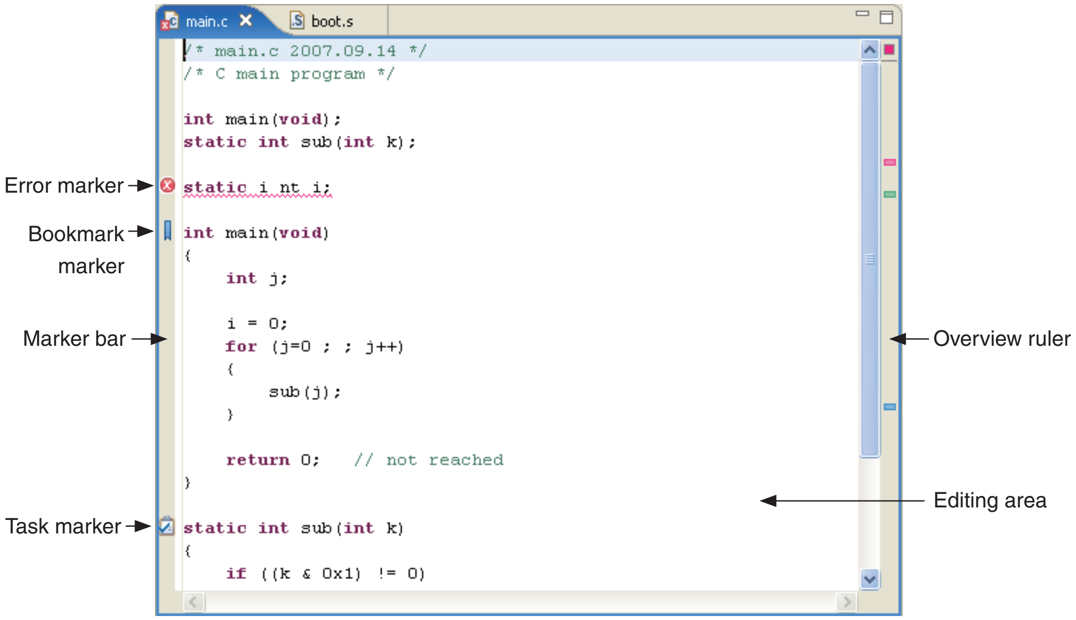
5.3.2 Window Toolbar



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 GNU33 Project]
New C/C++ Source Folder	= [File] > [New] > [Source Folder]
New C/C++ Source File	= [File] > [New] > [Source File]
New C++ Class	(Creates a new C++ class.)
External Tools (Run GDB33 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



```
/* main.c 2007.09.14 */
/* C main program */

int main(void);
static int sub(int k);

static i nt i;

int main(void)
{
    int j;

    i = 0;
    for (j=0 ; ; j++)
    {
        sub(j);
    }

    return 0;   // not reached
}

static int sub(int k)
{
    if ((k & 0x1) != 0)
```

This is the area of the editor where you edit source code. The **IDE** has an editor for C/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.

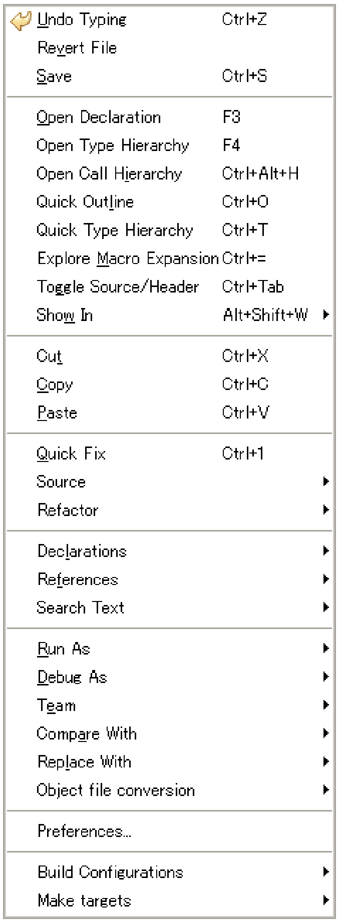
The marker bar on the left edge of the editor area shows the line in error and the markers indicating a bookmark, a line in which a task is set, etc. Hover the mouse pointer over a marker to display the contents of an error, the name of a bookmark, or a task explanation.

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 (for information on menu commands that are not described below, refer to the section that discusses the menu bar).


Undo Typing Ctrl+Z
Revert File
Save Ctrl+S
Open Declaration F3
Open Type Hierarchy F4
Open Call Hierarchy Ctrl+Alt+H
Quick Outline Ctrl+O
Quick Type Hierarchy Ctrl+T
Explore Macro Expansion Ctrl+=
Toggle Source/Header Ctrl+Tab
Show In Alt+Shift+W
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Quick Fix Ctrl+1
Source
Refactor
Declarations
References
Search Text
Run As
Debug As
Team
Compare With
Replace With
Object file conversion
Preferences...
Build Configurations
Make targets

Undo Typing = [Edit]>[Undo Typing]

Revert file

Returns the document currently being edited to the content saved immediately before in that file.

Save = [File]>[Save]

Open Declaration = [Navigate]>[Open Declaration]

Open Type Hierarchy = [Navigate]>[Open Type Hierarchy]

Open Call Hierarchy = [Navigate]>[Open Call Hierarchy]

Quick Outline = [Navigate]>[Quick Outline]

Toggle Source/Header = [Navigate]>[Toggle Source/Header]

Show in

Activates the view ([C/C++ Projects], [Navigator], [Project Explorer], or [Outline]) selected in the submenu and displays the file currently being edited. (C editor only)

Cut = [Edit]>[Cut]

Copy = [Edit]>[Copy]

Paste = [Edit]>[Paste]

Quick Fix = [Edit]>[Quick Fix]

Source

Displays the submenu for editing the line at which the cursor is currently positioned.

Comment/Uncomment

Changes the line at which the cursor is currently positioned to a comment line or ordinary source line ("//" added to or deleted from the beginning of the line). (C editor only)

Add Block Comment

Changes the currently selected string or line to a comment by enclosing with a set of "/*" and "*/." (C editor only)

Remove Block Comment

Delete a set of "/*" and "*/" from the currently selected string or line to change it to an ordinary source line. (C editor only)

Shift Right = [Edit]>[Shift Right]
Shift Left = [Edit]>[Shift Left]

Correct Indentation

Aligns the indent of the line being edited.

Format = [Edit]>[Format]
Add Include = [Edit]>[Add Include]
Content assist = [Edit]>[Content Assist] (C editor only)

Refactor

Rename...

Changes the name of the selected type, function, or member in all locations, including other locations in the source. The submenu displays the [Undo...] and [Redo...] commands, which enable cancellation and re-execution of the operation.

Declarations

Searches for the location of the declaration of the string (e.g., function name or variable name) selected in the editing area within the range selected in the submenu (workspace, current project, specified working set). (C editor only)

References

Searches for the location that references the string (e.g., function name or variable name) selected in the editing area within the range selected in the submenu (workspace, current project, specified working set). (C editor only)

Search Text

Searches for the string selected in the editing area within the range selected in the submenu (workspace, current project, current file, specified working set). (C editor only)

Preferences...

Displays the [Preferences] dialog box for the editor.

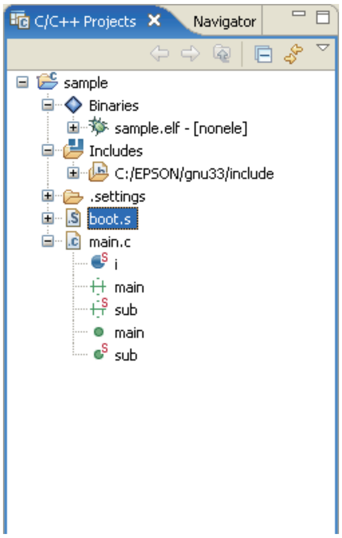
Build Configuration...

Defines the target to be selected by [Make Target...].

Make Target...

Selects a target and performs a build using **make.exe**.

5.3.4 [C/C++ Projects] View



Lists the projects present in the workspace along with the C/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/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
Binary container
Executable format file
Include container
Include folder
Header file
Source folder
C/C++ source file
Assembler source file
Text file, etc.
Object file
Include

Variable
Function
Class (class)
Structure (struct)
Member variable
Union (union)
Enumeration type (enum)
Enumerator
Function definition (prototype declaration)
Macro-definition
Type definition (typedef)
Namespace

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 immediately higher hierarchy.

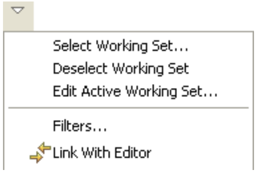
Collapse All

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...
Deselect Working Set
Edit Active Working Set...
Filters...
Link With Editor

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...

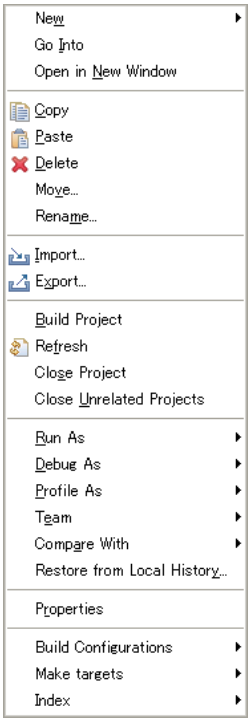
Specifies the type of file to be displayed.

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 (for information on menu commands not described below, refer to the section that discusses the menu bar).


New
Go Into
Open in New Window
Copy
Paste
Delete
Move...
Rename...
Import...
Export...
Build Project
Refresh
Close Project
Close Unrelated Projects
Run As
Debug As
Profile As
Team
Compare With
Restore from Local History...
Properties
Build Configurations
Make targets
Index

New = [File]>[New]

Open

Opens a selected file with an editor. (Effective when a file is selected)

Open With

Opens a selected file using the editor chosen from the following submenu. (Effective when a file is selected)

Go Into = [Navigate]>[Go Into]
(Effective when a project is selected)

Open in new Window = [Window]>[New Window]
(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...]

Build Project = [Project]>[Build Project]
(Effective when a project is selected)

Refresh = [File]>[Refresh]

Add Bookmark = [Edit]>[Add Bookmark]
(Effective when a project is selected)

Close Project = [Project]>[Close Project]
(Effective when a project is selected)

Close Unrelated Project

Closes projects unrelated to the one currently selected. (Effective when a project is selected)

Compare With

Compares the contents of two or three selected files.

Restore from Local History...

Restores files (e.g., those that have been deleted) to a project. (Effective when a project is selected)

Replace With (Effective when a file is selected)

Local History...

Replaces a selected file with the content previously saved (selected from history).

Previous from Local History

Replaces a selected file with the content saved immediately before.

Object file conversion (Effective when a file is selected)

Generate an S record file (Effective when an elf file is selected)

Converts a selected elf format object file to Motorola S3 format and generates a HEX file. This calls a command that executes "`objcopy -I elf32-little -O srec --srec-forceS3` <*filename*>`.elf` <*filename*>`.sa`."

Generate a raw binary file (Effective when an elf file is selected)

Removes debugging and other information from a selected elf format object file and generates a binary file. This calls a command that executes "`objcopy -I elf32-little -O binary` <*filename*>`.elf` <*filename*>`.bin`."

Properties

Displays a [Properties] dialog box that shows and allows changes in the properties of the current project.

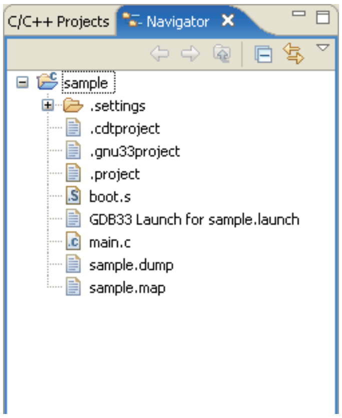
Build Configuration...

Defines the target to be selected by [Make Target...].

Make Target...

Selects a target and performs a build using **make.exe**.

5.3.5 [Navigator] View


C/C++ Projects
Navigator
sample
.settings
.cdtproject
.gnu33project
.project
boot.s
GDB33 Launch for sample.launch
main.c
sample.dump
sample.map

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.

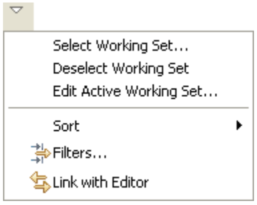
Collapse All

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...
Deselect Working Set
Edit Active Working Set...
Sort
Filters...
Link with Editor

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...

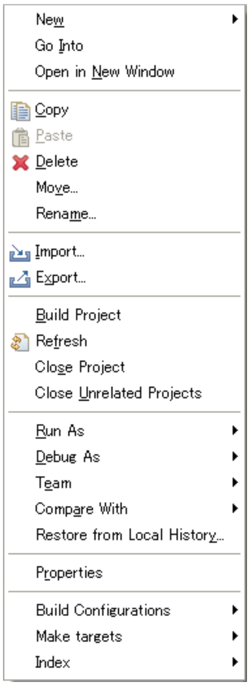
Specifies the type of file to be displayed.

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 (for information on menu commands not described below, refer to the section that discusses the menu bar).


New
Go Into
Open in New Window
Copy
Paste
Delete
Move...
Rename...
Import...
Export...
Build Project
Refresh
Close Project
Close Unrelated Projects
Run As
Debug As
Team
Compare With
Restore from Local History...
Properties
Build Configurations
Make targets
Index

New = [File]>[New]

Open

Opens a selected file with an editor. (Effective when a file is selected)

Open With

Opens a selected file using the editor chosen from the following submenu. (Effective when a file is selected)

Go Into = [Navigate]>[Go Into]
(Effective when a project is selected)

Open in new Window = [Window]>[New Window]
(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...]

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 Project

Closes projects unrelated to the one currently selected. (Effective when a project is selected)

Compare With

Compares the contents of two or three selected files.

Restore from Local History...

Restores files (e.g., those that have been deleted) to a project. (Effective when a project is selected)

Replace With (Effective when a file is selected)

Local History...

Replaces a selected file with the content previously saved (selected from history).

Previous from Local History

Replaces a selected file with the content saved immediately before.

Object file conversion (Effective when a file is selected)

Generate an S record file (Effective when an elf file is selected)

Converts a selected elf format object file to Motorola S3 format and generates a HEX file. This calls a command that executes "`objcopy -I elf32-little -O srec --srec-forceS3` *<filename>*`.elf` *<filename>*`.sa`."

Generate a raw binary file (Effective when an elf file is selected)

Removes debugging and other information from a selected elf format object file and generates a binary file. This calls a command that executes "`objcopy -I elf32-little -O binary <`*filename*`>.elf <`*filename*`>.bin.`"

Properties

Displays a [Properties] dialog box that shows and allows changes in the properties of the current project.

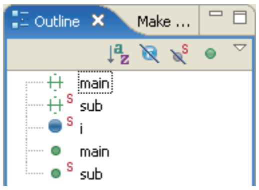
Build Configuration...

Defines the target to be selected by [Make Target...].

Make Target...

Selects a target and performs a build using **make.exe**.

5.3.6 [Outline] View


Outline
Make ...
main
sub
i
main
sub

Shows the functions, classes, and global variables that are written in the C/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.

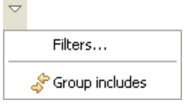
Hide Static Members

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...
Group includes

Filters...

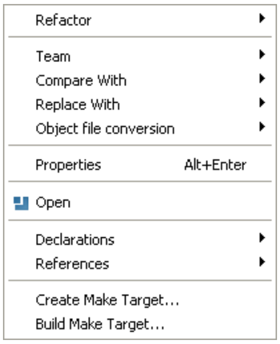
Specifies the items to be displayed in the view.

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
Team
Compare With
Replace With
Object file conversion
Properties Alt+Enter
Open
Declarations
References
Create Make Target...
Build Make Target...

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).

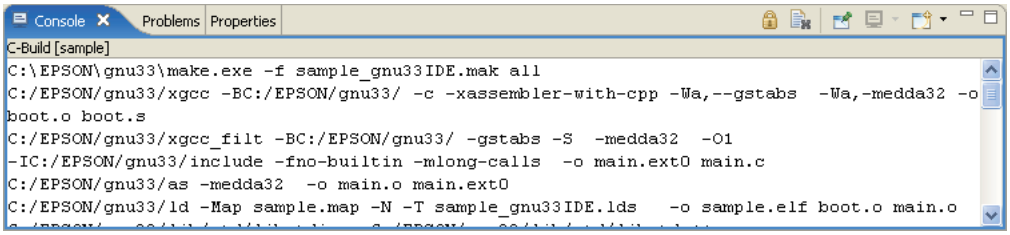
Create Make Target...

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:\EPSON\gnu33\make.exe -f sample_gnu33IDE.mak all
C:/EPSON/gnu33/xgcc -BC:/EPSON/gnu33/ -c -xassembler-with-cpp -Wa,--gstabs  -Wa,-medda32 -o
boot.o boot.s
C:/EPSON/gnu33/xgcc_filt -BC:/EPSON/gnu33/ -gstabs -S  -medda32  -O1
-IC:/EPSON/gnu33/include -fno-builtin -mlong-calls  -o main.ext0 main.c
C:/EPSON/gnu33/as -medda32  -o main.o main.ext0
C:/EPSON/gnu33/ld -Map sample.map -N -T sample_gnu33IDE.lds   -o sample.elf boot.o main.o
```

Displays the executed command line or the messages output by the GNU33 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

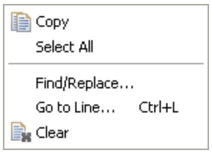
This button is displayed in a debugger startup console, etc. It closes the active console.

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

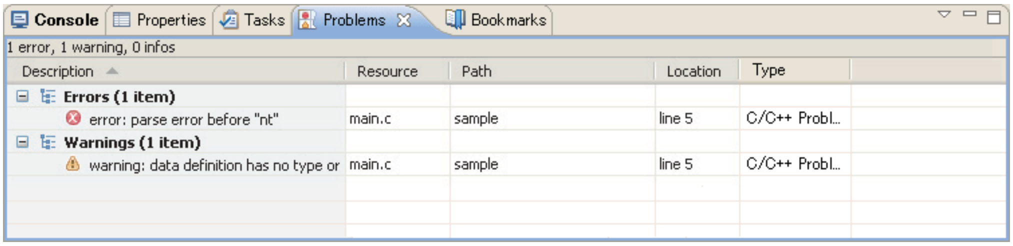
Clears the contents displayed.

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



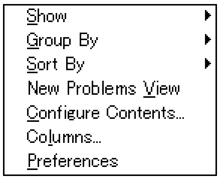
Console
Properties
Tasks
Problems
Bookmarks

1 error, 1 warning, 0 infos

Description	Resource	Path	Location	Type
Errors (1 item)				
error: parse error before "nt"	main.c	sample	line 5	C/C++ Probl...
Warnings (1 item)				
warning: data definition has no type or	main.c	sample	line 5	C/C++ Probl...

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.

View menu



Show
Group By
Sort By
New Problems View
Configure Contents...
Columns...
Preferences

Show

Selects a filter to be applied.

Group By

Selects a target for grouping of errors.

Sort By

Select the items in the list you wish to prioritize over others when sorting the list. Selecting [Ascending] sorts and arranges items in ascending order. Deselecting [Ascending] sorts and arranges items in descending order.

New Problems View

Creates a new problems view.

Configure Contents...

Creates and edits filter settings.
The conditions set here restrict the errors to be displayed and the maximum allowable number. For filter settings, refer to “5.10.7 Filters.”

Columns...

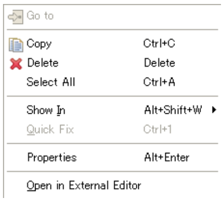
Sets the sequence of items (rows) to be displayed.

Preferences...

Sets the maximum number of errors to be displayed and the display/hide setting for items (rows).

Context menu

Right-click in the view to display the context menu shown below.



Go to
Copy Ctrl+C
Delete Delete
Select All Ctrl+A
Show In Alt+Shift+W
Quick Fix Ctrl+1
Properties Alt+Enter
Open in External Editor

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]

Properties

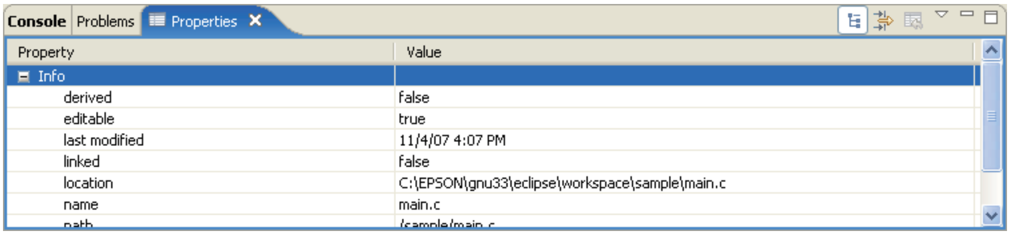
Displays information on the error currently selected.

Open in External Editor

If an external editor has been set according to the procedure described in Section 5.5.10, "Launching External Editor by Specifying Line Number", you can jump to the error-generating line by using the context menu.
For details, refer to Section 5.5.10, "Launching External Editor by Specifying Line Number".

(For the menu commands not specifically discussed here, refer to the description of the menu bar.)

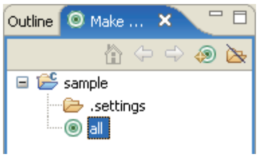
5.3.9 [Properties] View



Property	Value
⊟ Info	
derived	false
editable	true
last modified	11/4/07 4:07 PM
linked	false
location	C:\EPSON\gnu33\eclipse\workspace\sample\main.c
name	main.c
path	/sample/main.c

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


Outline
Make ...
sample
.settings
all

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

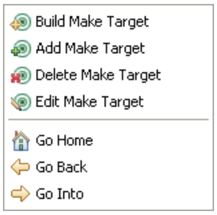
Executes a make process on a selected target.

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
Add Make Target
Delete Make Target
Edit Make Target
Go Home
Go Back
Go Into

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.

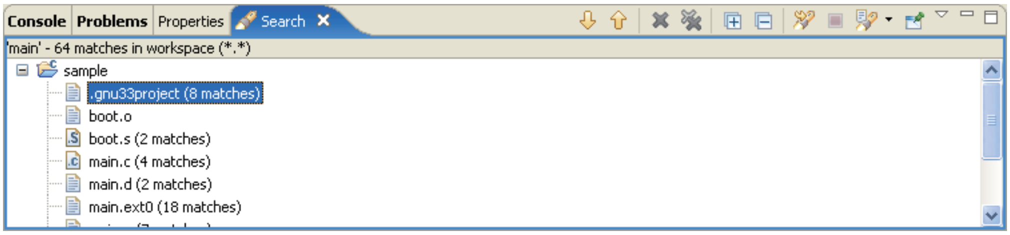
Go Back

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


Console
Problems
Properties
Search
'main' - 64 matches in workspace (*.*)
sample
.gnu33project (8 matches)
boot.o
boot.s (2 matches)
main.c (4 matches)
main.d (2 matches)
main.ext0 (18 matches)

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.

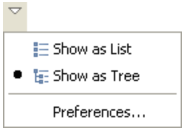
Show Previous Searches

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


Show as List
● Show as Tree
Preferences...

Show as List

Shows the search results in a non-hierarchical flat layout.

Show as Tree

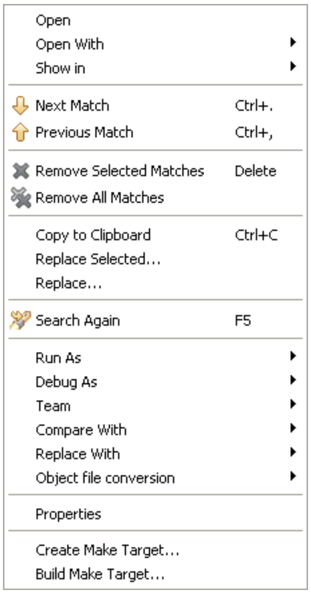
Shows the search results in hierarchically structured mode.

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
Open With
Show in
Next Match Ctrl+.
Previous Match Ctrl+,
Remove Selected Matches Delete
Remove All Matches
Copy to Clipboard Ctrl+C
Replace Selected...
Replace...
Search Again F5
Run As
Debug As
Team
Compare With
Replace With
Object file conversion
Properties
Create Make Target...
Build Make Target...

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/C++ editor (when C/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/C++ editor (when C/C++ source is selected) or assembly editor (when assembler source is selected)

Default Editor

C/C++ editor (when C/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 -I elf32-little -O srec --srec-forceS3` *<filename>*`.elf` *<filename>*`.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 "`objcopy -I elf32-little -O binary` *<filename>*`.elf` *<filename>*`.bin`".

Properties

Displays information on the currently selected occurrence of search string.

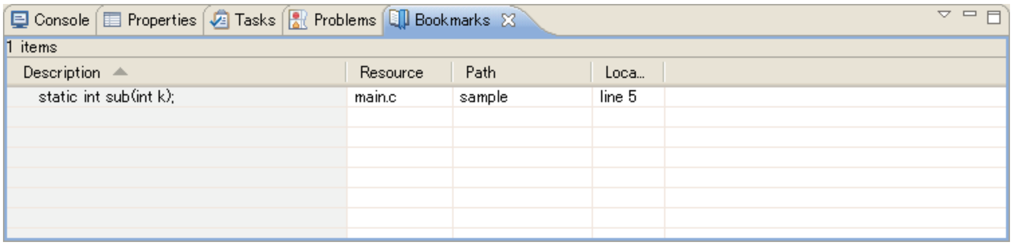
Create Make Target...

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

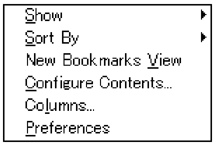

Console
Properties
Tasks
Problems
Bookmarks

1 items

Description	Resource	Path	Loca...
static int sub(int k);	main.c	sample	line 5

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.)

View menu


Show
Sort By
New Bookmarks View
Configure Contents...
Columns...
Preferences

Show

Selects a filter to be applied.

Sort By

Select the items in the list you wish to prioritize over other items when sorting the list. Selecting [Ascending] sorts and arranges items in ascending order. Deselecting [Ascending] sorts and arranges items in descending order.

New Bookmarks View

Creates a new bookmarks view.

Configure Contents...

Creates and edits filter settings.
The conditions set here restrict the bookmarks to be displayed and the maximum allowable number. For filter settings, refer to 5.10.7, “Filters”.

Columns...

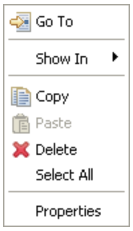
Sets the sequence of items (rows) to be displayed.

Preferences...

Sets the maximum number of bookmarks to be displayed and the display/ hide setting for items (rows).

Context menu

Right-click in the view to display the context menu shown below.


Go To
Show In
Copy
Paste
Delete
Select All
Properties

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]

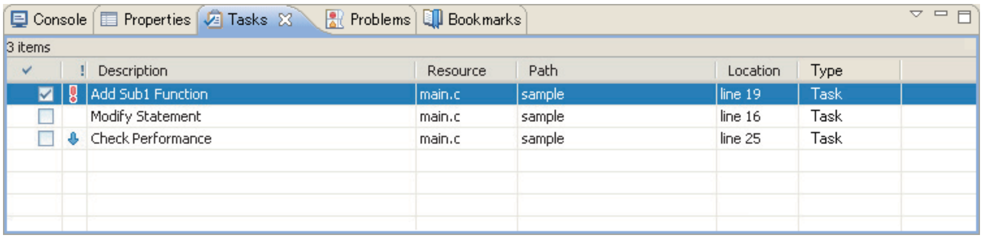
Select All = [Edit] > [Select All]

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

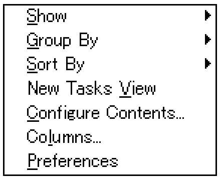

Console
Properties
Tasks
Problems
Bookmarks

3 items

✓	!	Description	Resource	Path	Location	Type
☑	!	Add Sub1 Function	main.c	sample	line 19	Task
☐		Modify Statement	main.c	sample	line 16	Task
☐	↓	Check Performance	main.c	sample	line 25	Task

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.

View menu


Show
Group By
Sort By
New Tasks View
Configure Contents...
Columns...
Preferences

Show

Selects a filter to be applied.

Group By

Selects a target for grouping of tasks.

Sort By

Select the items in the list you wish to prioritize over other items when sorting the list. Selecting [Ascending] sorts and arranges items in ascending order. Deselecting [Ascending] sorts and arranges items in descending order.

New Tasks View

Creates a new tasks view.

Configure Contents...

Creates and edits filter settings.
The conditions set here restrict the tasks to be displayed and the maximum allowable number. For filter settings, refer to 5.10.7, “Filters”.

Columns...

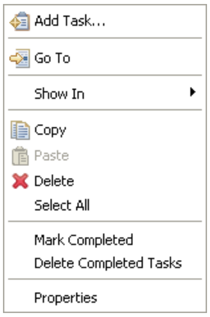
Sets the sequence of items (rows) to be displayed.

Preferences...

Sets the maximum number of tasks to be displayed and the display/hide setting for items (rows).

Context menu

Right-click in the view to display the context menu shown below.


Add Task...
Go To
Show In
Copy
Paste
Delete
Select All
Mark Completed
Delete Completed Tasks
Properties

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

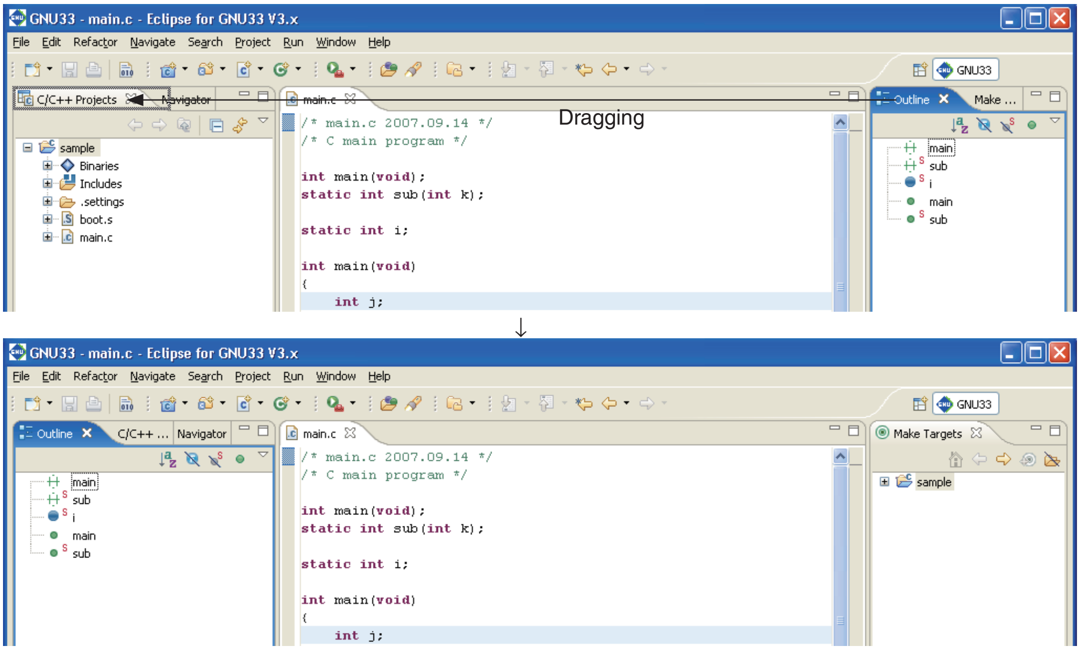
The displayed view is closed by clicking the ☒ button on the tab. When all views in one pane are closed, the pane itself goes out.

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.


GNU33 - main.c - Eclipse for GNU33 V3.x
File Edit Refactor Navigate Search Project Run Window Help
GNU33
C/C++ Projects
Navigator
main.c
Outline
Make ...
Dragging
sample
Binaries
Includes
.settings
boot.s
main.c
main
sub
i
main
sub
↓
GNU33 - main.c - Eclipse for GNU33 V3.x
File Edit Refactor Navigate Search Project Run Window Help
GNU33
Outline
C/C++ ...
Navigator
main.c
Make Targets
sample
main
sub
i
main
sub

```
/* main.c 2007.09.14 */
/* C main program */

int main(void);
static int sub(int k);

static int i;

int main(void)
{
    int j;
```

The size of any pane can also be changed by dragging the boundary border of the pane.

Maximizing a view

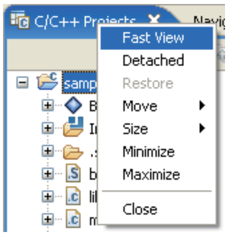
When you double-click the tab of a view, a pane including the selected view is expanded to the size of the **IDE** window, with other views hidden behind it. When you double-click the tab of a maximized view, the view reverts to its original size. Each pane has a maximize button at the upper right corner. 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.

☐ Maximize button

▭ Minimize button

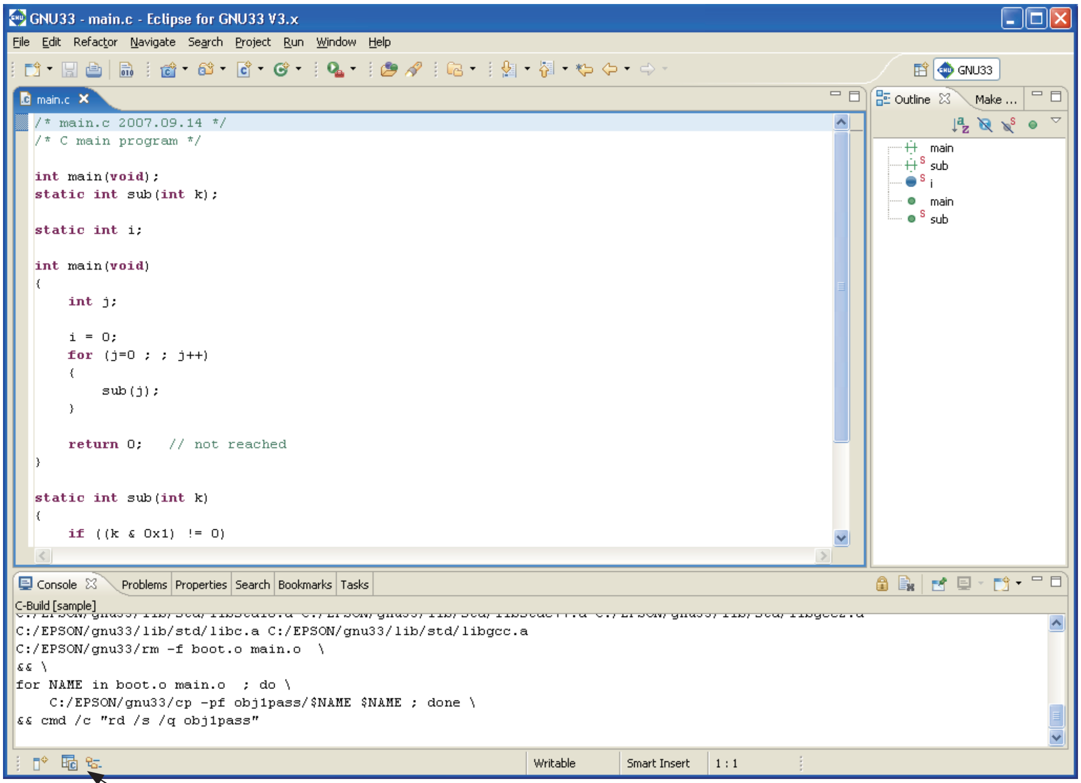
Fast view

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.


C/C++ Projects
Navig
Fast View
Detached
Restore
Move
Size
Minimize
Maximize
Close
samp

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.

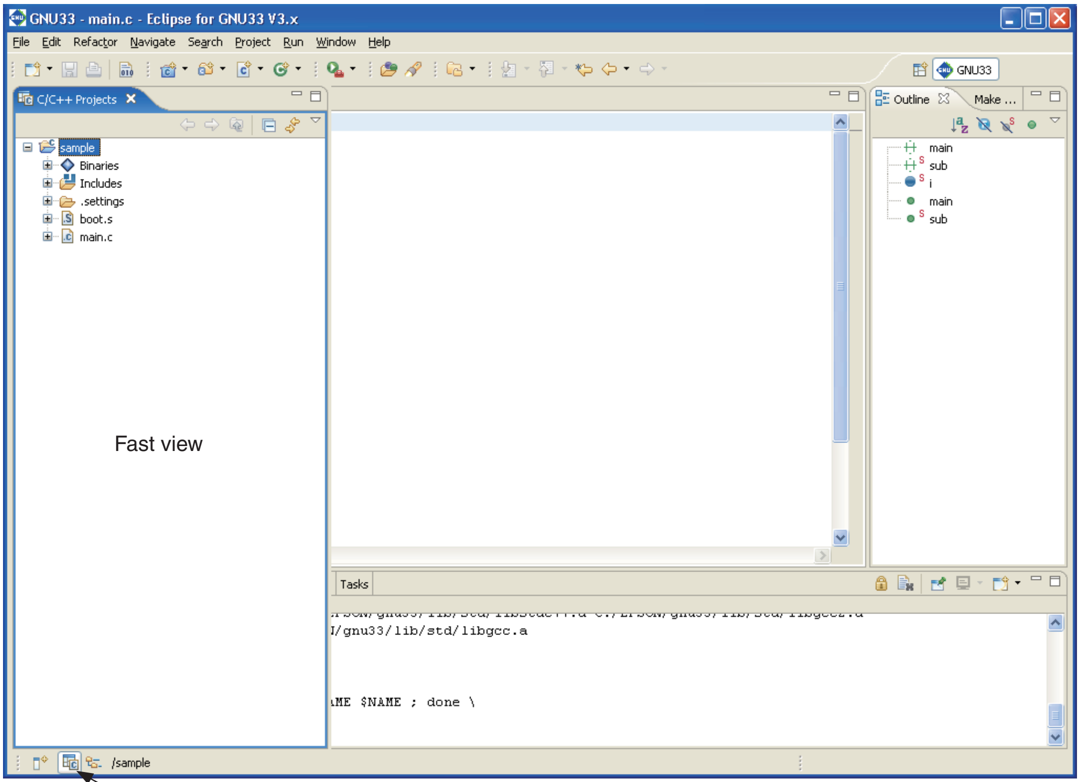

GNU33 - main.c - Eclipse for GNU33 V3.x
File
Edit
Refactor
Navigate
Search
Project
Run
Window
Help
GNU33
main.c
Outline
Make ...
main
sub
i
main
sub
Console
Problems
Properties
Search
Bookmarks
Tasks
C-Build [sample]
Writable
Smart Insert
1 : 1

```
/* main.c 2007.09.14 */
/* C main program */

int main(void);
static int sub(int k);

static int i;

int main(void)
{
    int j;

    i = 0;
    for (j=0 ; ; j++)
    {
        sub(j);
    }

    return 0;    // not reached
}

static int sub(int k)
{
    if ((k & 0x1) != 0)
```

```
C:/EPSON/gnu33/lib/std/libc.a C:/EPSON/gnu33/lib/std/libgcc.a
C:/EPSON/gnu33/rm -f boot.o main.o  \
&& \
for NAME in boot.o main.o  ; do \
    C:/EPSON/gnu33/cp -pf obj1pass/$NAME $NAME ; done \
&& cmd /c "rd /s /q obj1pass"
```

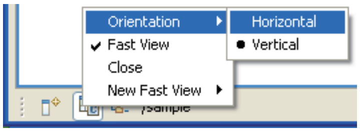
Fast view area

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.

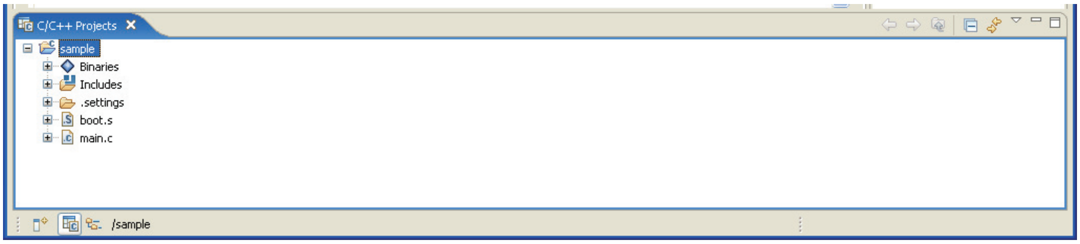

GNU33 - main.c - Eclipse for GNU33 V3.x
File Edit Refactor Navigate Search Project Run Window Help
GNU33
C/C++ Projects
sample
Binaries
Includes
.settings
boot.s
main.c
Fast view
Outline
Make ...
main
sub
i
main
sub
Tasks
/sample

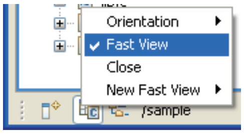
```
I/gnu33/lib/std/libgcc.a

AME $NAME ; done \
```

Click the icon to open or close a fast view


Orientation
Fast View
Close
New Fast View
Horizontal
Vertical
/sample

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.

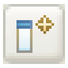

C/C++ Projects
sample
Binaries
Includes
.settings
boot.s
main.c
/sample


Orientation
Fast View
Close
New Fast View
/sample

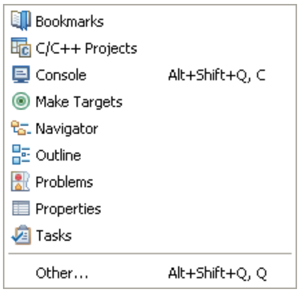
To restore a fast view to normal view, select [Fast View] from the context menu for the fast-view icon.

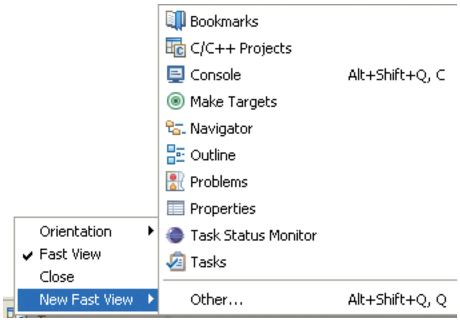
Select [Close] from this context menu to close the view and remove the icon.

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


Bookmarks
C/C++ Projects
Console Alt+Shift+Q, C
Make Targets
Navigator
Outline
Problems
Properties
Tasks
Other... Alt+Shift+Q, Q


Orientation
Fast View
Close
New Fast View
Bookmarks
C/C++ Projects
Console Alt+Shift+Q, C
Make Targets
Navigator
Outline
Problems
Properties
Task Status Monitor
Tasks
Other... Alt+Shift+Q, Q

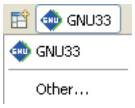
To restore view layout to default settings

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 "GNU33" is defined with the view configuration and layout described in the preceding sections.


GNU33
GNU33
Other...

The word "GNU33" shown to the right of the perspective shortcut icon indicates that the GNU33 perspective is currently selected. Although the **IDE** allows you to switch perspectives, use only the default "GNU33" 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 (`.cproject`, `.gnu33project`, and `.project`) are generated in a project directory and are updated from time to time by the **IDE**.

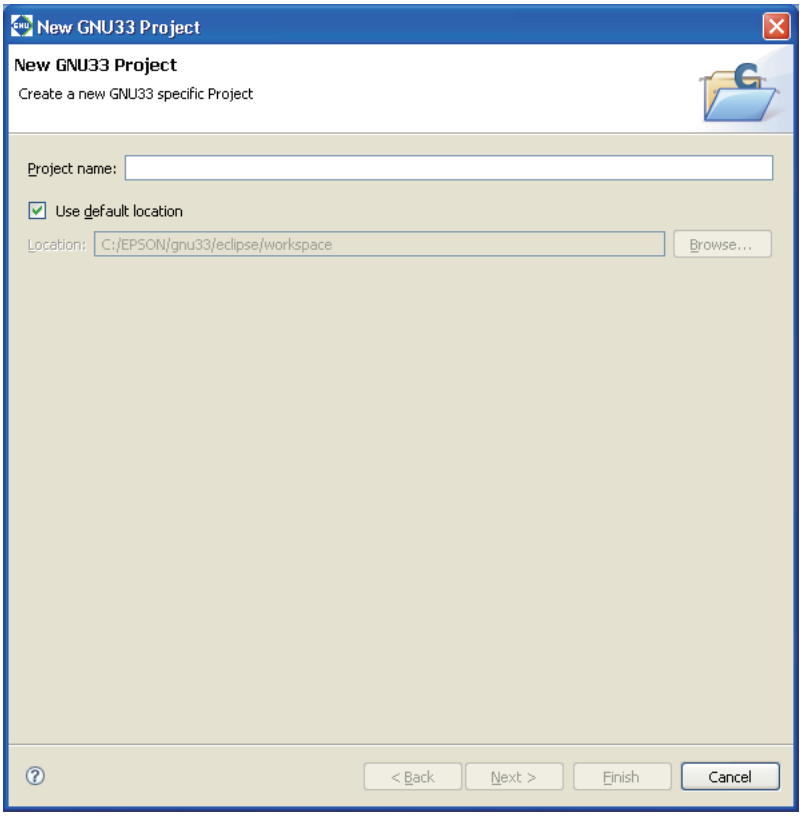
Note: These project management files which reside in the project directory must not be edited, moved, or deleted except when you manipulate them in the **IDE**. Attempting to do so will prevent you from restarting the project.

5.4.2 Creating a New Project

Application development by the **IDE** starts with creating a new project:

(1) Launch the [New GNU33 Project] wizard by one of the following methods.
- Select [New] > [New GNU33 Project] from the [File] menu.
- Select [New GNU33 Project] from the [New] shortcut in the toolbar.
- Select [New] > [New GNU33 Project] from the context menu for the [C/C++ Projects] or [Navigator] view.

The wizard will start, displaying the dialog box shown below.


New GNU33 Project
New GNU33 Project
Create a new GNU33 specific Project
Project name:
☑ Use default location
Location: C:/EPSON/gnu33/eclipse/workspace
Browse...
< Back
Next >
Finish
Cancel

(2) Enter a project name in the [Project name:] text box.

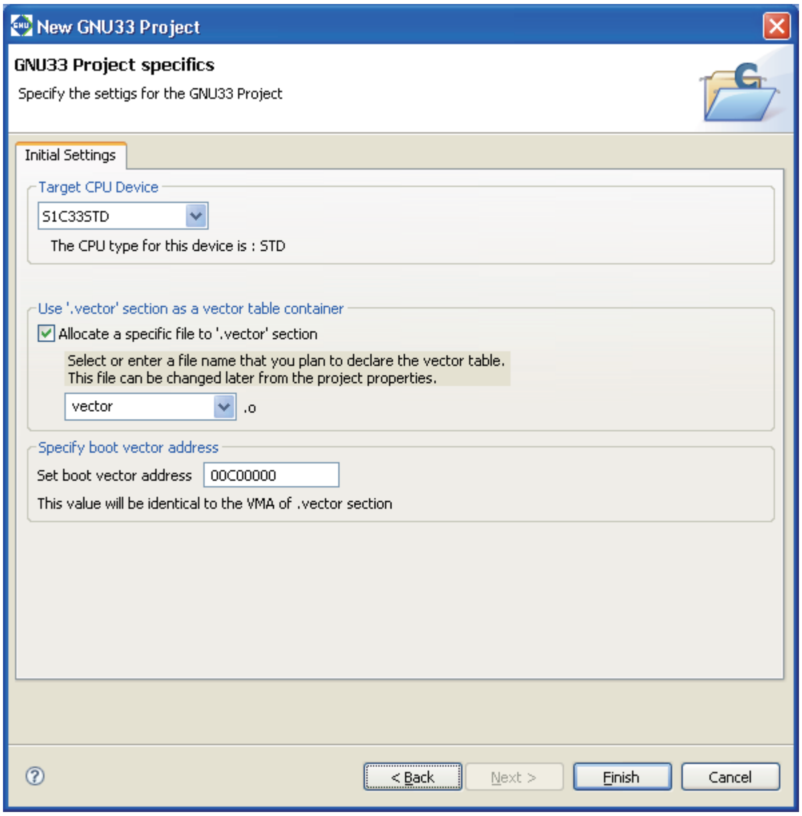
Notes:
- Make sure the project name you enter is 100 characters or less.
- Only single-byte alphanumeric characters and underscores may be used for project names.

(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.



New GNU33 Project

GNU33 Project specifics

Specify the settigs for the GNU33 Project

Initial Settings

Target CPU Device: S1C33STD

The CPU type for this device is : STD

Use '.vector' section as a vector table container

☑ Allocate a specific file to '.vector' section

Select or enter a file name that you plan to declare the vector table.
This file can be changed later from the project properties.

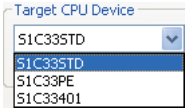
vector .o

Specify boot vector address

Set boot vector address 00C00000

This value will be identical to the VMA of .vector section

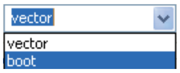
(5) From the [Target CPU Device] combo box, select the type of C33 Core incorporated by the target CPU:



S1C33STD: When the target processor is a device with the C33 STD Core embedded
S1C33PE: When the target processor is a device with the C33 PE Core embedded
S1C33401: When the target processor is the S1C33401

You can switch target CPUs later. (Refer to Section 5.7.1, "Setting the CPU Type".)

(6) 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 5.7.7, "Editing a Linker Script".)

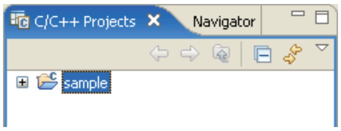
(7) Specify the boot vector address. The default value is "00C00000" when "S1C33STD" or "S1C33PE" is selected as the target CPU or "20000000" when "S1C33401" is selected. The value set here will be used as the parameter for the TTBR setting command that will be written in the debugger startup command 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.

(8) Click the [Finish] button.

The [New GNU33 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.

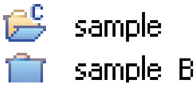
In the [C/C++ Projects] or [Navigator] view, the project will be displayed along with a directory icon similar to the one shown below.
Example: Project created with the name "sample"



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



sample ← Open project

sample B ← Closed project

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] form the [Project] menu.
  - Select [Open Project] form the context menu for the [C/C++ Projects] or [Navigator] view.

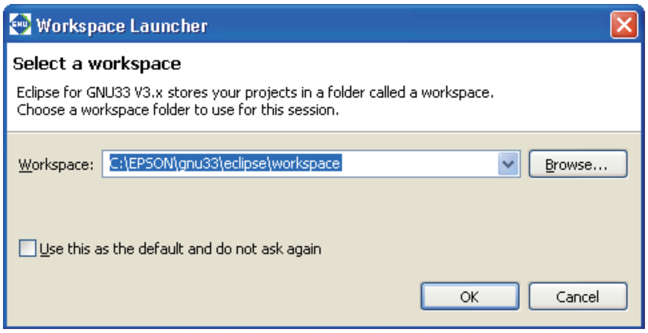
This works only for currently closed projects displayed in the [C/C++ Projects] or [Navigator] view. To open a project present in a directory outside the current workspace, switch the workspace to the directory in which the project is saved (see Section 5.4.4). Or import an existing project you want 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.



Workspace Launcher

Select a workspace

Eclipse for GNU33 V3.x stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\EPSON\gnu33\eclipse\workspace

☐ Use this as the default and do not ask again

(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.

* Do not specify the project directory (directory containing .project file) as a workspace directory. Doing so may result in failures with project imports (when [Copy projects into workspace] is selected).

(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.

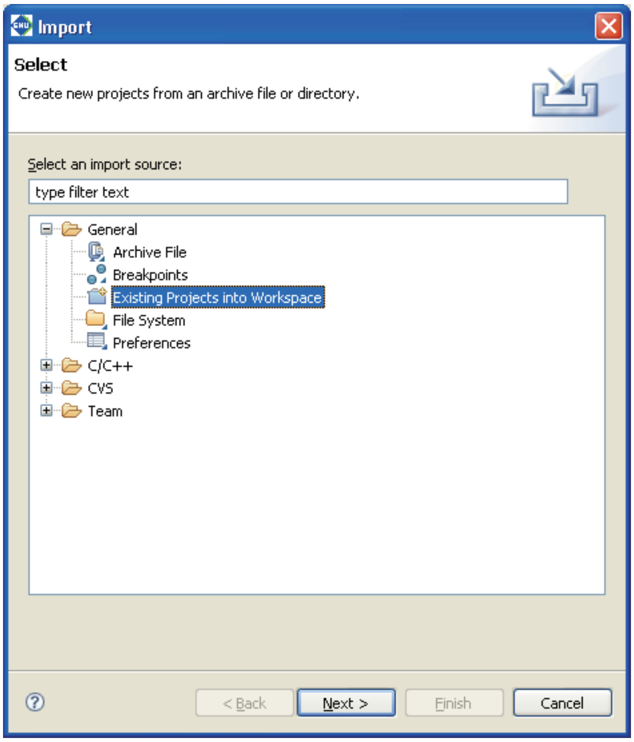
Specifying a nonexistent directory in (3) will create a new instance of your specified directory.

5.4.5 Importing an Existing Project

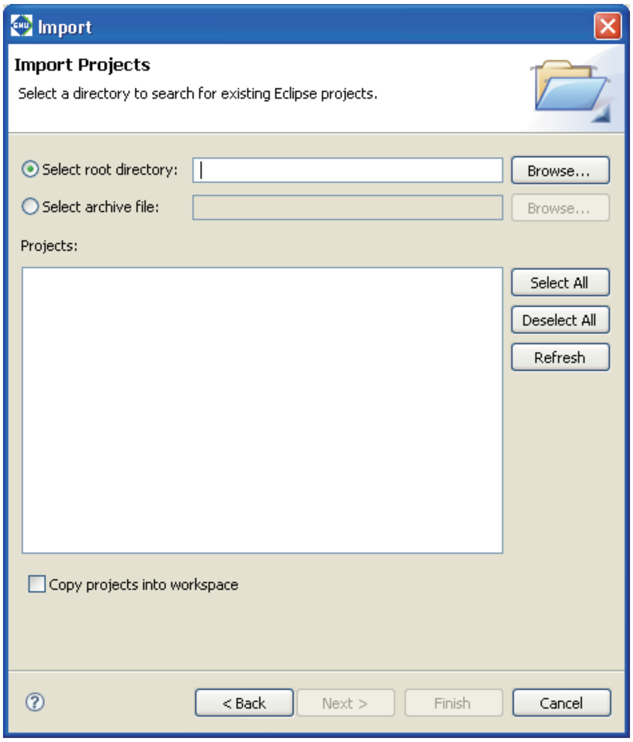
This section describes how to import an existing project into the current workspace.
Follow the import procedure given below to continue developing a project created in the IDE of an older version (V3.1 or later).
The import procedure is described below.

(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.


Import
Select
Create new projects from an archive file or directory.
Select an import source:
type filter text
General
Archive File
Breakpoints
Existing Projects into Workspace
File System
Preferences
C/C++
CVS
Team
< Back
Next >
Finish
Cancel

(2) Select [Existing Projects into Workspace] from the list and click [Next>].



Import Projects

Select a directory to search for existing Eclipse projects.

○ Select root directory: |

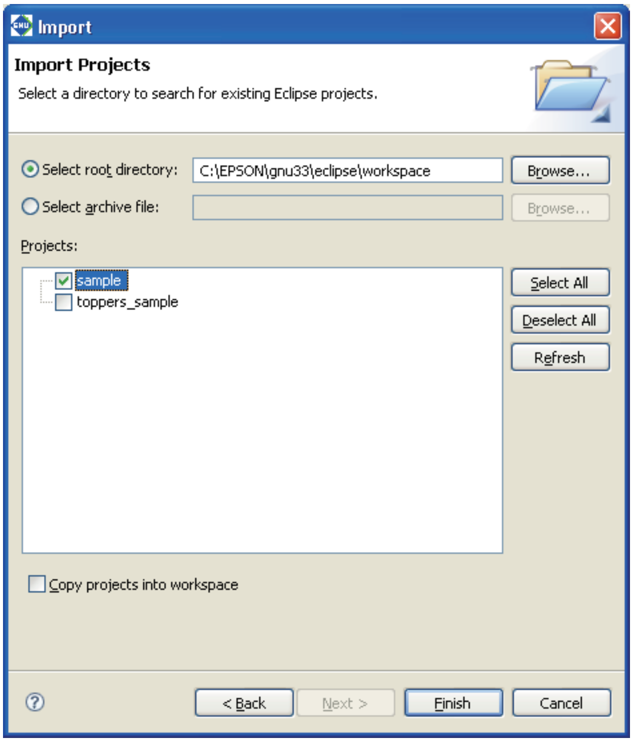
○ Select archive file:

Projects:

☐ Copy projects into workspace

Import
Browse...
Browse...
Select All
Deselect All
Refresh
< Back
Next >
Finish
Cancel

(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.



Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: C:\EPSON\gnu33\eclipse\workspace

○ Select archive file:

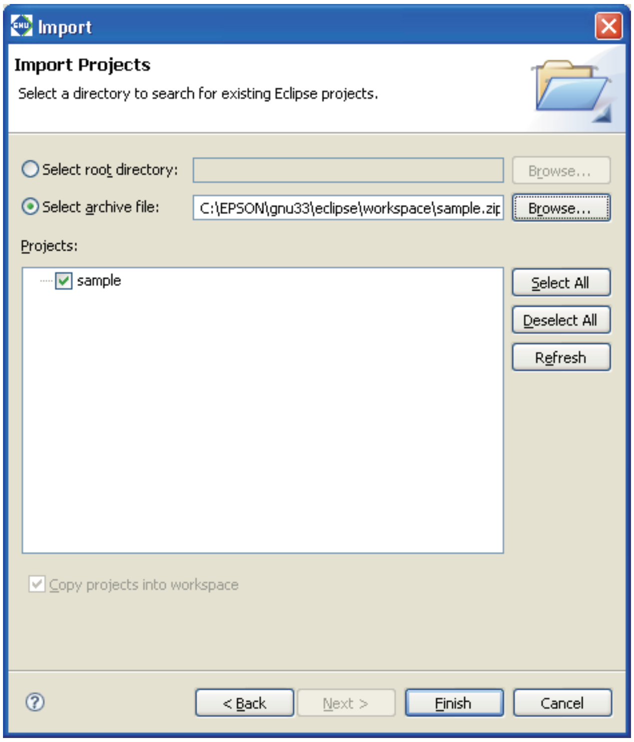
Projects:

- ☑ sample
- ☐ toppers_sample

☐ Copy projects into workspace

Import
Browse...
Browse...
Select All
Deselect All
Refresh
< Back
Next >
Finish
Cancel

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.


Import Projects

Select a directory to search for existing Eclipse projects.

○ Select root directory:

● Select archive file: C:\EPSON\gnu33\eclipse\workspace\sample.zi

Projects:

☑ sample

☑ Copy projects into workspace

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.

* Do not specify the project directory (directory containing .project file) as a workspace directory. Doing so may result in failures with project imports (when [Copy projects into workspace] is selected).

(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.

Automatic updates of the project file

When you import a project created in an older version of the IDE, the project file (.project/.cproject/.gnu33project/) is automatically updated to one compatible with the current version.
Note that .cdtproject files used by projects from an older version will be replaced by .cproject files.
A .cproject file is generated during a project import, and the contents of the .cdtproject file will be transferred automatically to the newly generated .cproject file.

Directory structure and resource location

If all resources are stored together in a project folder, the project can be copied to any location. The project can then be built at the copied destination with no further revisions.
Even if external files are referenced from inside the original project folder, there is no need for modification as long as the directory structure is the same. However, if makefiles and other files are prepared externally and are not automatically generated by the IDE, as explained in Tutorial 2, the paths specified in these files may need to be corrected.
No problems will arise with the standard libraries and include directories as long as the tools are installed in the same directory (e.g., C:\EPSON\gnu33). If you use an IDE with a different tool directory, the user library and include directory must be changed as necessary using [GNU33 Build Options] in the [Properties] dialog box of the project.

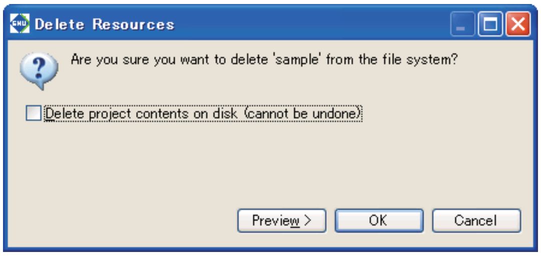
5.4.6 Deleting a Project

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 [Delete Resources] dialog box is displayed.



Delete Resources

Are you sure you want to delete 'sample' from the file system?

☐ Delete project contents on disk (cannot be undone)

∗ To delete all directories and files in the file system along with the project

(3) Deselect the [Delete project contents on disk] checkbox.
In this case, the project displayed in the [C/C++ Projects] and [Navigator] views disappears, but the files remain. If you import the same project (refer to Section 5.4.5), you can continue working on it as a project.

∗ To delete all directories and files in the file system along with the project

(3') Select the [Delete project contents on disk] checkbox.

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 [OK] button to delete. To cancel, click the [Cancel] 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.

If a project cannot be deleted on Windows Vista

When an attempt to delete a project is made on Windows Vista by selecting [Delete project contents on disk] in the [Delete Resources] dialog box, an error message, “An exception has been caught while processing the refactoring ‘Delete Resource’” may appear and disallow the deletion of a project folder in some cases.
This error occurs when conime.exe (enabling Japanese input in command prompt) launches using the project folder as the current directory during a project build and continues running after the project is built, disallowing the deletion of the project folder.
When this happens, use the Task Manager to end the conime.exe process or reboot the PC, and then delete the project folder.

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*>`_gnu33IDE.cmd`)
  The previous file <*old project name*>`_gnu33IDE.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*>`_gnu33IDE.mak`)*
  - Linker script file (<*project name*>`_gnu33IDE.lds`)*
  - Parameter file (<*project name*>`_gnu33IDE.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 Configurations] 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>*`_gnu33IDE.cmd` and paste them in to the [Properties] > [GNU33 GDB Commands] page as necessary.
- To place a copy of the original project in the same workspace, please paste the project first and then rename it.

  Example: To place Project and Project2 in the same workspace
  1. Copy project “Project”.
  2. Paste the project with the name “projectTemp.”
  3. Rename “projectTemp” to “Project2.”

  This procedure lets you place an original project “Project” and a new project “Project2” in the same workspace.

- Note that the automatic correction of the project contents shown above will not be performed when the project is copied and pasted using the context menu in the [C/C++ Projects] or [Navigator] view.

5.4.8 Resource Manipulation in a Project

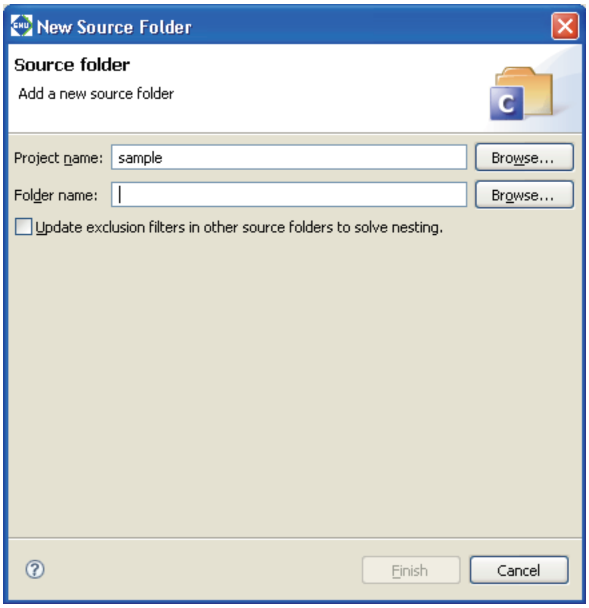
This section describes the operations to create resources, as well as importing, copying, moving, or deleting resources that are possible in the [C/C++ Projects] or [Navigator] view.

Creating a new source directory

You can create a new source directory in a project.
The source files used in the project must be stored in a source folder. The **IDE** automatically includes the source files located in a source folder into the make file and updates the linker script file so that the object files generated from the source files will be linked. The project folder is always handled as a source folder.
A procedure to create a source folder is described below.

(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.


New Source Folder

Source folder

Add a new source folder

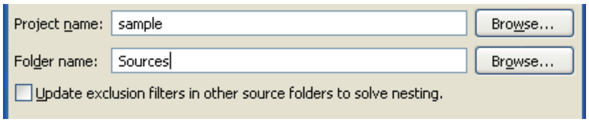
Project name: sample

Folder name:

☐ Update exclusion filters in other source folders to solve nesting.

(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.


Project name: sample

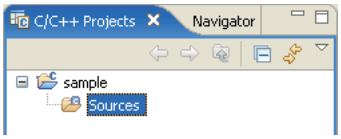
Folder name: Sources

☐ Update exclusion filters in other source folders to solve nesting.

If source files are already imported into the root directory of the project, select the [Update exclusion filters in other source folders to solve nesting.] check box. When a source folder is created without selecting the check box, the source files must be imported again.
If a source file is placed in the project folder, be sure to select this checkbox (it is selected by default).

(4) Click the [Finish] button. To cancel, click the [Cancel] button.

The directory you created is displayed.


C/C++ Projects
Navigator
sample
Sources

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.

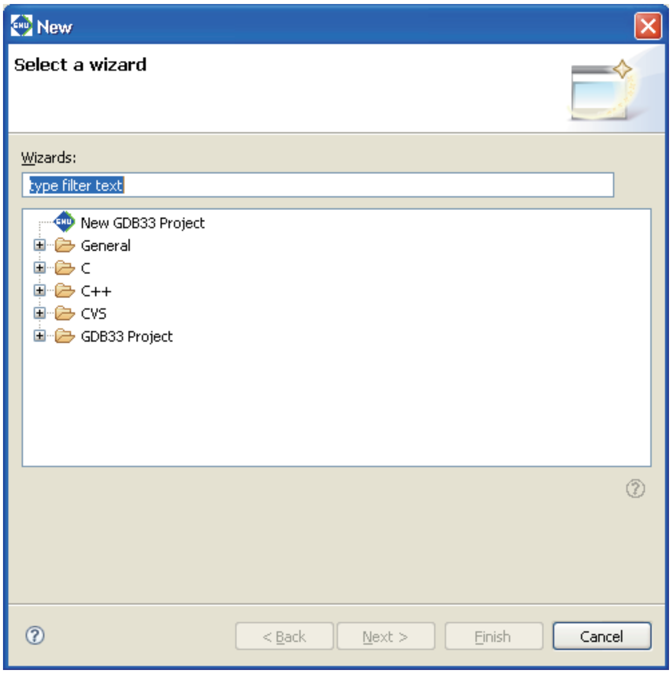
Creating a new directory for general-purpose use

You can create a new directory in a project or in the internal directory of a project:

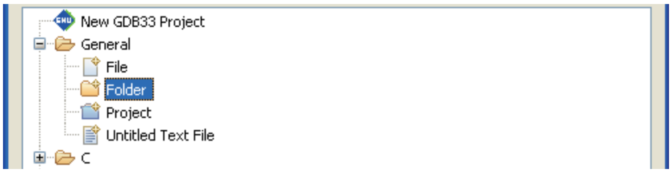
(1) Perform one of the following operations.

- 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.

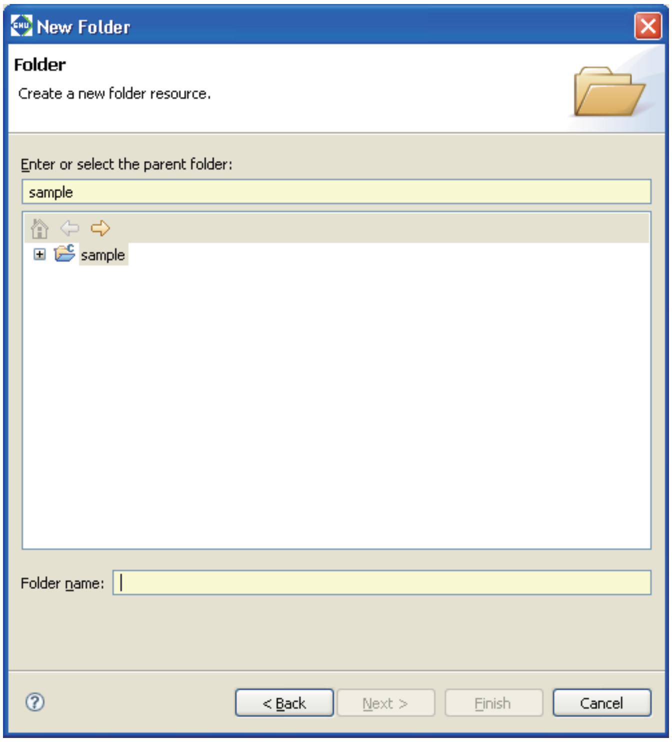
The [New] dialog box is displayed.


New
Select a wizard
Wizards:
type filter text
New GDB33 Project
General
C
C++
CVS
GDB33 Project
< Back
Next >
Finish
Cancel

(2) Select [Folder] from the [General] tree list.


New GDB33 Project
General
File
Folder
Project
Untitled Text File
C

(3) Click [Next >].


New Folder
Folder
Create a new folder resource.
< Back
Next >
Finish
Cancel

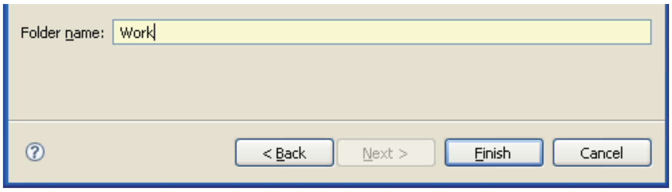
Enter or select the parent folder: sample

sample

Folder name:

(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.

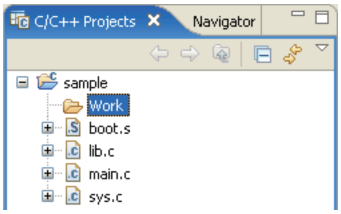
(5) Enter the name of the directory you want to create in the [Folder name:] text box.


< Back
Next >
Finish
Cancel

Folder name: Work

(6) Click the [Finish] button. To cancel, click the [Cancel] button.

The directory you created is displayed.


C/C++ Projects
Navigator
sample
Work
boot.s
lib.c
main.c
sys.c

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.

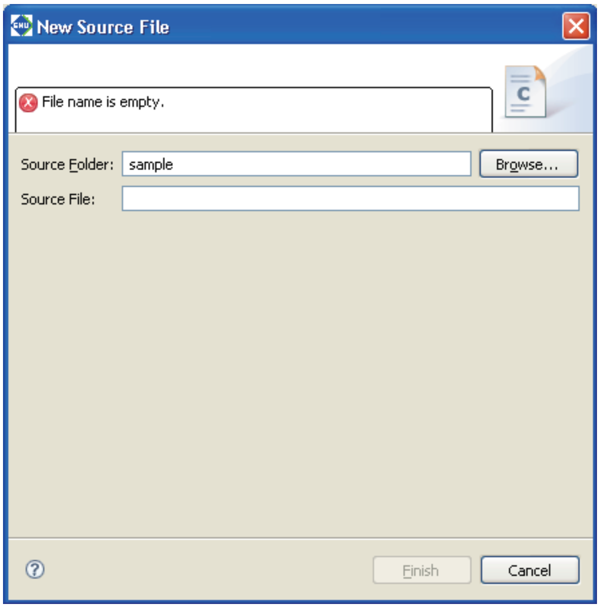
Creating a new source file/header file

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.

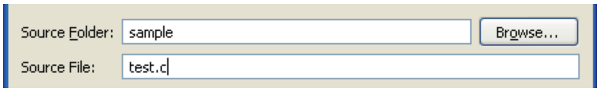


File name is empty.

Source Folder: sample

Source File:

(2) Enter the name of the file you want to create in the [Source File:] (or [Header File:]) text box.

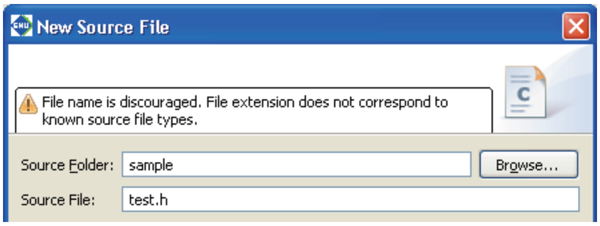


Source Folder: sample

Source File: test.c

Enter the file extension as ".c" to create a C source file, ".cpp" 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.



File name is discouraged. File extension does not correspond to known source file types.

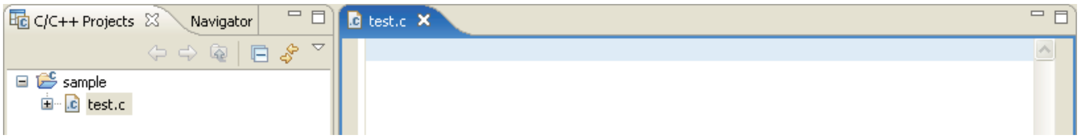
Source Folder: sample

Source File: test.h

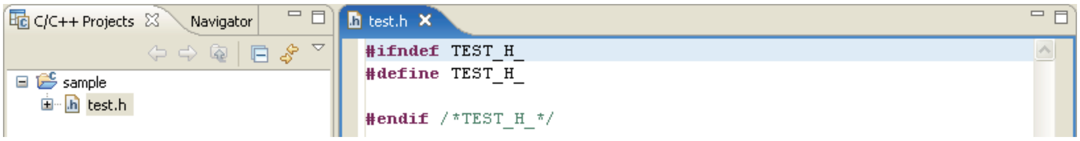
The current project name is entered in [Source Folder:]. When creating the file in another directory, enter the path or select the directory using the [Browse...] button.

(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.


C/C++ Projects
Navigator
sample
test.c
test.c

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.


C/C++ Projects
Navigator
sample
test.h
test.h

```
#ifndef TEST_H_
#define TEST_H_

#endif /*TEST_H_*/
```

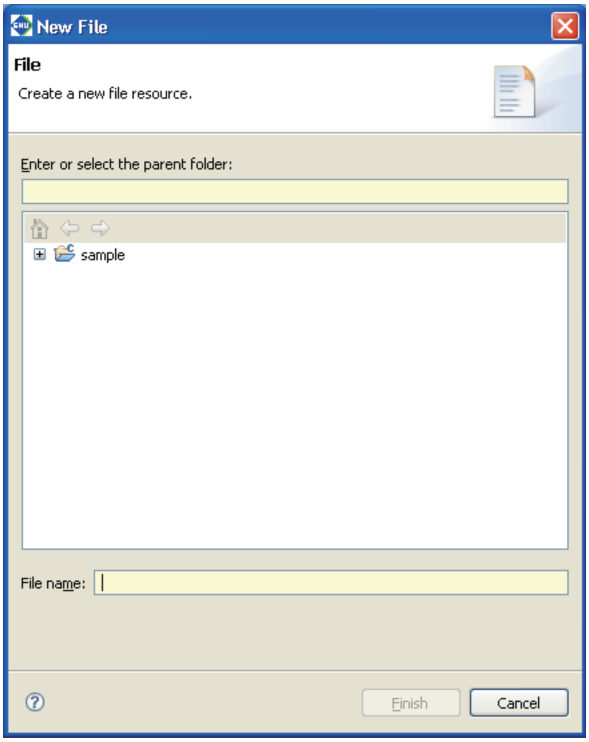
Creating a new general-purpose text file

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.

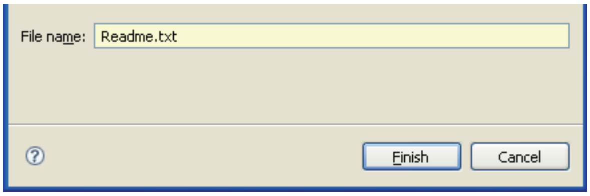
- Select [New] > [File] from the [File] menu.
- Select [New] > [File] from the context menu for the [C/C++ Projects] or [Navigator] view.
- Select [File] from the [New] shortcut in the toolbar.
- Select [File] from the [New C/C++ Source File] shortcut in the toolbar.

The [New File] dialog box is displayed.


New File
File
Create a new file resource.
Enter or select the parent folder:
sample
File name:
Finish
Cancel

(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.

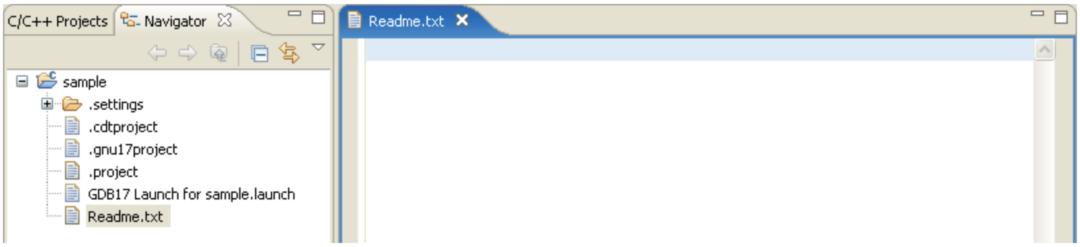


File name: Readme.txt

Finish
Cancel

(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.



C/C++ Projects
Navigator
sample
.settings
.cdtproject
.gnu17project
.project
GDB17 Launch for sample.launch
Readme.txt
Readme.txt

* 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.

Creating a new class

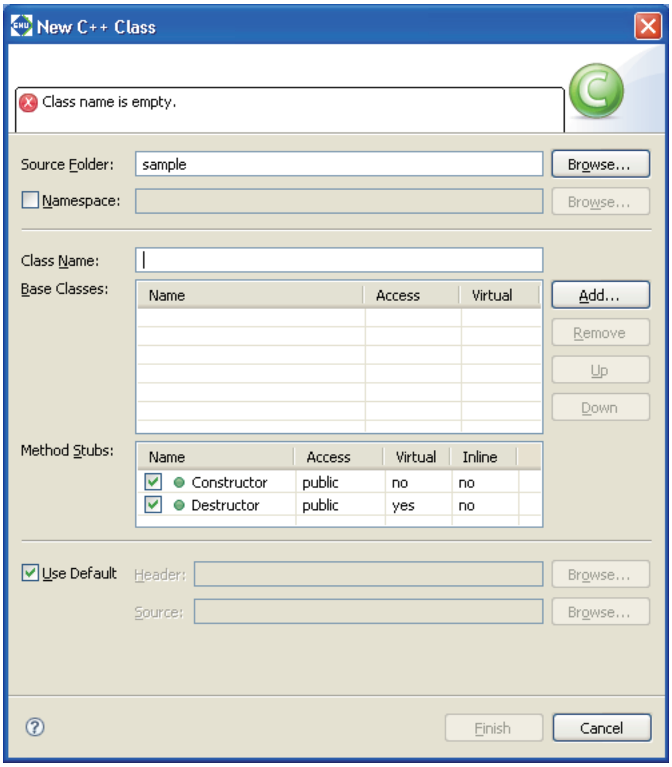
You can generate the source and header files in a project that have a new C++ class written in each:

(1) Select the project in which you want to create a new class.

(2) Perform one of the following operations.

- Click the [New C++ Class] button in the toolbar.
- Select [New] > [Other...] from the [File] menu and select [C++] > [Class] from the list on the dialog box that appears.
- Select [New] > [Other...] from the context menu for the [C/C++ Projects] or [Navigator] view and select [C++] > [Class] from the list on the dialog box that appears.
- Select [New] > [Other...] from the [New] shortcut in the toolbar and select [C++] > [Class] from the list on the dialog box that appears.

The [New C++ Class] dialog box is displayed (see Section 5.10.3 for details).



New C++ Class

Class name is empty.

Source Folder: sample

☐ Namespace:

Class Name:

Base Classes:

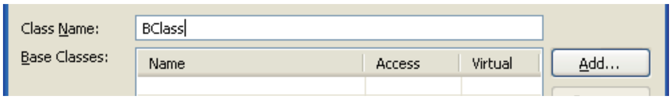
Name	Access	Virtual

Method Stubs:

Name	Access	Virtual	Inline
☑ Constructor	public	no	no
☑ Destructor	public	yes	no

☑ Use Default Header: Source:

(3) Enter the name of the class you want to create in the [Class Name:] text box.



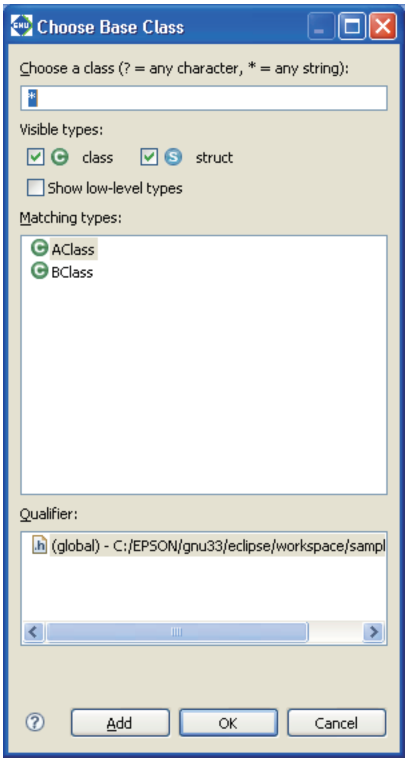
Class Name: BClass

Base Classes:

Name	Access	Virtual

(4) To inherit from another class, enter the base class in the [Base Classes:] text box by selecting one from the [Choose Base Class] dialog box displayed by clicking the [Add...] button (see Section 5.10.4 for details).

Note: To select a base class or a namespace, [Fast C/C++ Indexer] must be selected on the [C/C++ Indexer] page of the project property in advance.



Choose a class (? = any character, * = any string): *

☑ class ☑ struct

☐ Show low-level types

AClass
BClass

(global) - C:/EPSON/gnu33/eclipse/workspace/sampl

The classes and their structures currently defined in the current project will be listed in the [Matching types:] column of the [Choose Base Class] dialog box, so select the base class from the list and click the [OK] button.

If the list has many entries, deselect the [struct] check box to narrow down the contents of the list to class.
Furthermore, enter part of characters comprising the class name and a wildcard (?: character, *: string) in the [Choose a class (? = any character, * = any string):] text box. That way you can display only the classes that include the specified character.

(5) Make other settings as necessary (see Section 5.10.3 for details).

(6) Click the [Finish] button. To cancel, click the [Cancel] button.

Example: Create BClass from AClass as the base class

Header file (BClass.h)

```
#ifndef BCLASS_H
#define BCLASS_H                   ... Macro to prevent double-inclusions

#include "AClass.h"                ... Include a declaration of the base class

class BClass : public AClass{      ... Declares a class
public:

     BClass();                     ... Constructor
     virtual ~BClass();            ... Virtual destructor
};

#endif // BCLASS_H
```

The contents written in the header file are a macro to prevent double-inclusions and a class declaration (constructor and virtual destructor) according to the [Method Stubs:] setting (see Section 5.10.3 for details).

Source file (BClass.cpp)

```
#include "BClass.h"          ... Include a header file

AClass::BClass()             ... Define a constructor
{}
AClass::~BClass()            ... Define a destructor
{}
```

The contents written in the source file are a header file inclusion, a constructor definition, and a destructor definition according to the [Method Stubs:] setting (see Section 5.10.3 for details).

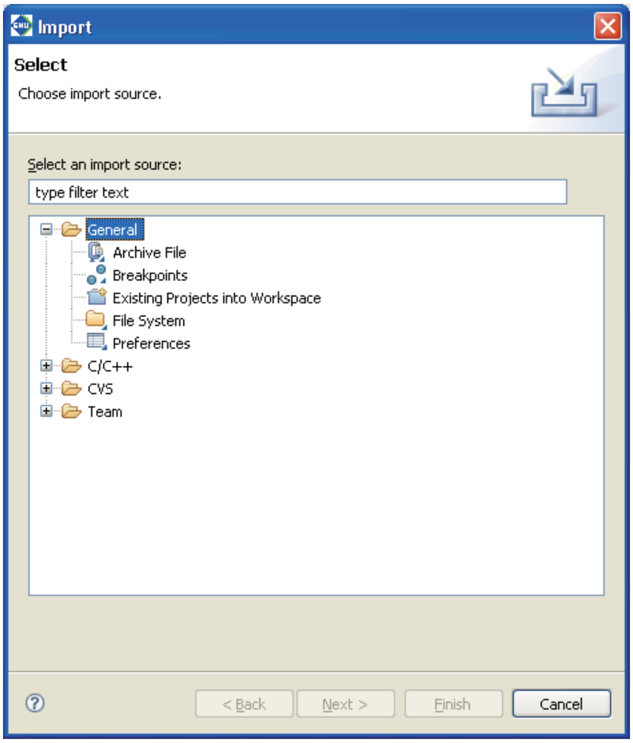
Importing an existing file or directory

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.

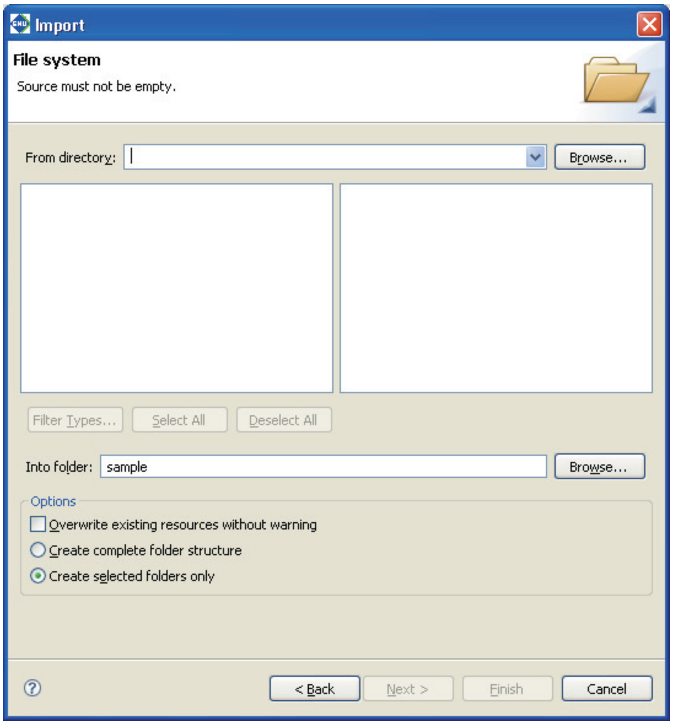
(1) From within the [C/C++ Projects] or [Navigator] view, click on the project or directory into which you want to import a file/directory.

(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.


Import
Select
Choose import source.
Select an import source:
type filter text
General
Archive File
Breakpoints
Existing Projects into Workspace
File System
Preferences
C/C++
CVS
Team
< Back
Next >
Finish
Cancel

(3) Select [General] > [File System] from the list and click [Next>].



Import

File system

Source must not be empty.

From directory:

Into folder: sample

Options

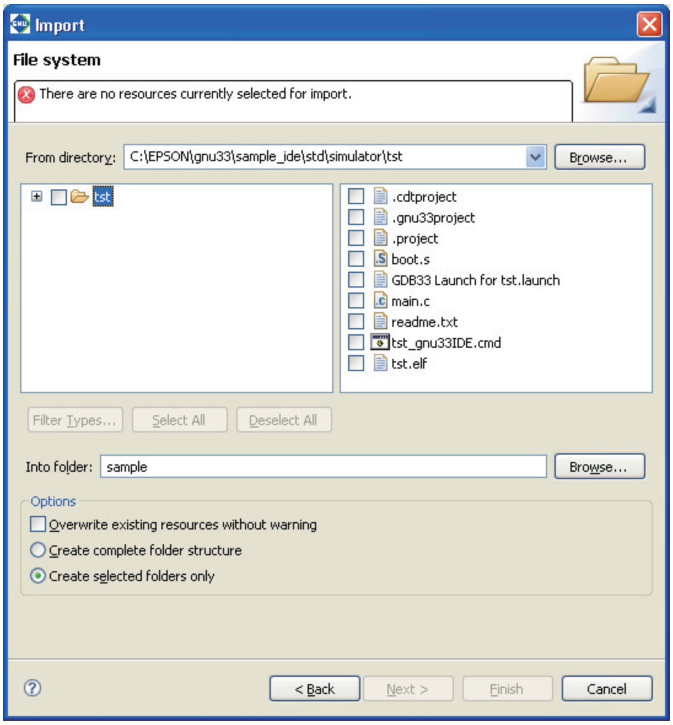
☐ Overwrite existing resources without warning

○ Create complete folder structure

● Create selected folders only

Browse...
Filter Types...
Select All
Deselect All
Browse...
< Back
Next >
Finish
Cancel

(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.



Import

File system

There are no resources currently selected for import.

From directory: C:\EPSON\gnu33\sample_ide\std\simulator\tst

☐ tst

☐ .cdtproject
☐ .gnu33project
☐ .project
☐ boot.s
☐ GDB33 Launch for tst.launch
☐ main.c
☐ readme.txt
☐ tst_gnu33IDE.cmd
☐ tst.elf

Into folder: sample

Options

☐ Overwrite existing resources without warning

○ Create complete folder structure

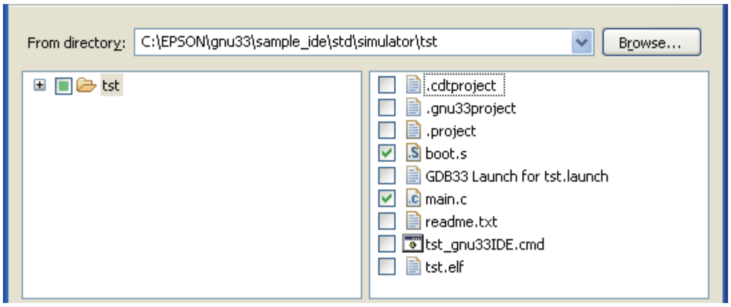
● Create selected folders only

Browse...
Filter Types...
Select All
Deselect All
Browse...
< Back
Next >
Finish
Cancel

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.

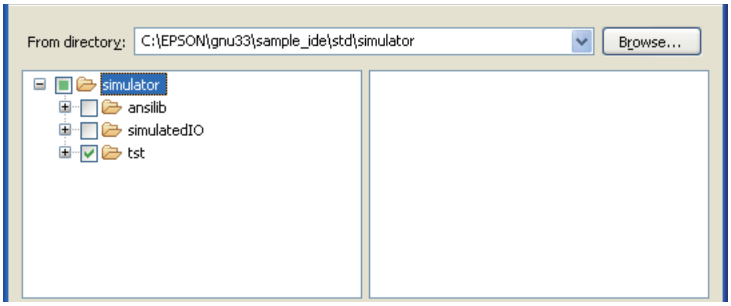
(5) To import a file, select the file you want to import from the list box to the right (indicated by a check mark when selected).

Example: Importing a file


From directory:
C:\EPSON\gnu33\sample_ide\std\simulator\tst
Browse...
tst
☐ .cdtproject
☐ .gnu33project
☐ .project
☑ boot.s
☐ GDB33 Launch for tst.launch
☑ main.c
☐ readme.txt
☐ tst_gnu33IDE.cmd
☐ tst.elf

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


From directory:
C:\EPSON\gnu33\sample_ide\std\simulator
Browse...
simulator
☐ ansilib
☐ simulatedIO
☑ tst

(6) Click the [Finish] button.

The imported file/directory is displayed in the [C/C++ Projects] or [Navigator] view.

Refer to Section 5.10.5 for information on the [Import > File system] dialog box and other controls.

Specifying a source folder

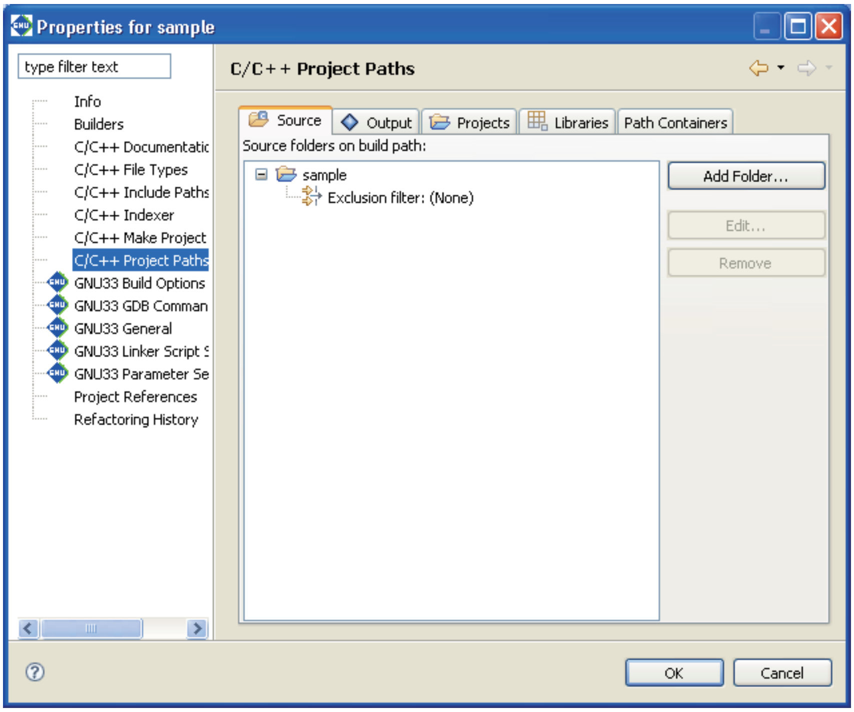
The **IDE** writes the source files located directly below the project directory to a makefile, thereby specifying those files as targets to be assembled/compiled. In their initial default status, source files located in the internal directory of a project are not written in a makefile. However, you 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.

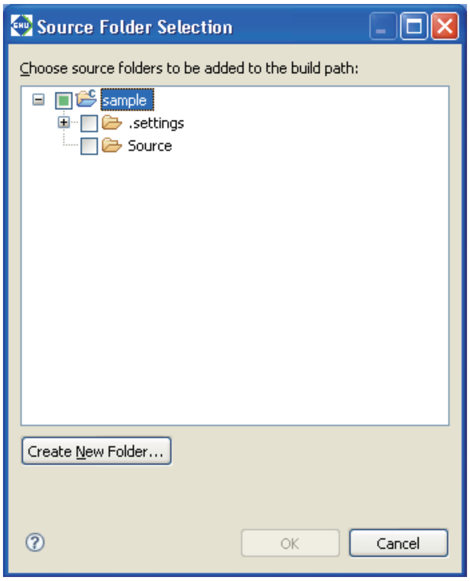
This will display the [Properties] dialog box.

(3) Select [C/C++ Project Paths] from the properties list, then click the [Source] tab.

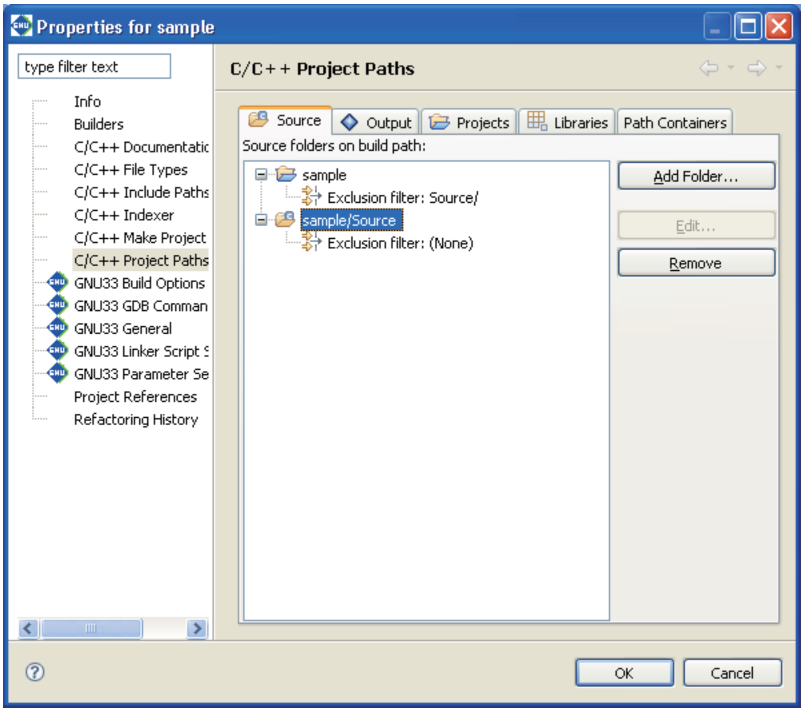

Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
C/C++ Project Paths
Source
Output
Projects
Libraries
Path Containers
Source folders on build path:
sample
Exclusion filter: (None)
Add Folder...
Edit...
Remove
OK
Cancel

By default, project directories are already registered.

(4) Click the [Add Folder...] button.
This displays the [Source Folder Selection] dialog box.


Source Folder Selection
Choose source folders to be added to the build path:
sample
.settings
Source
Create New Folder...
OK
Cancel

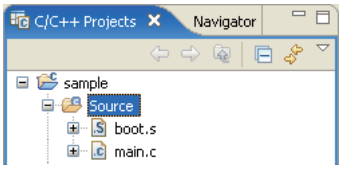
(5) Select the directory you want to register (indicated by a check mark when selected) and click the [OK] button.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
C/C++ Project Paths
Source
Output
Projects
Libraries
Path Containers
Source folders on build path:
sample
Exclusion filter: Source/
sample/Source
Exclusion filter: (None)
Add Folder...
Edit...
Remove
OK
Cancel

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.


C/C++ Projects
Navigator
sample
Source
boot.s
main.c

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.

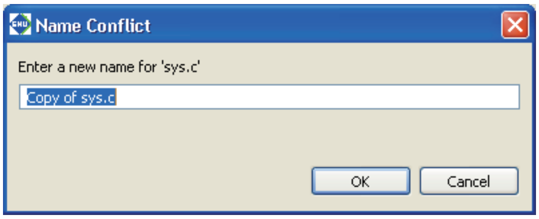
This copies the selected resource to the clipboard.

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.



Enter a new name for 'sys.c'

Copy of sys.c

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.

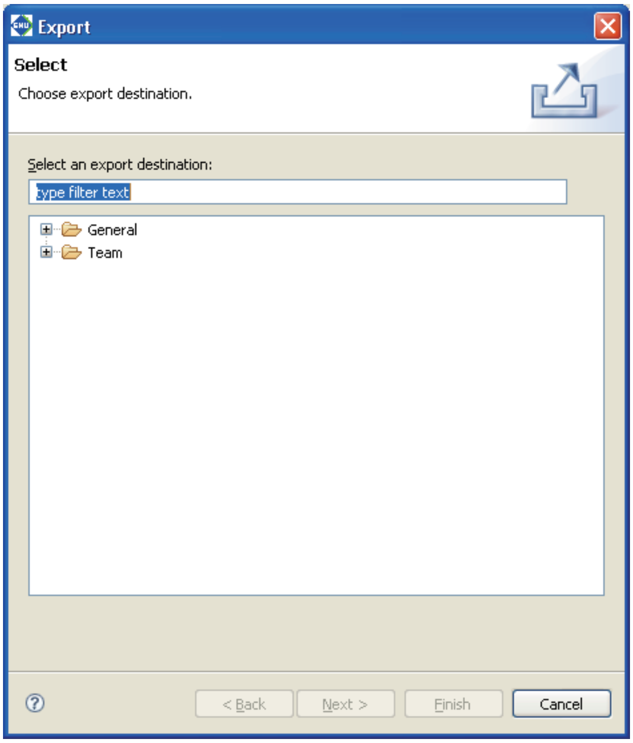
Exporting a file or directory

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.



Export

Select

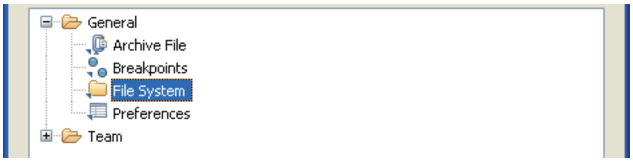
Choose export destination.

Select an export destination:

type filter text

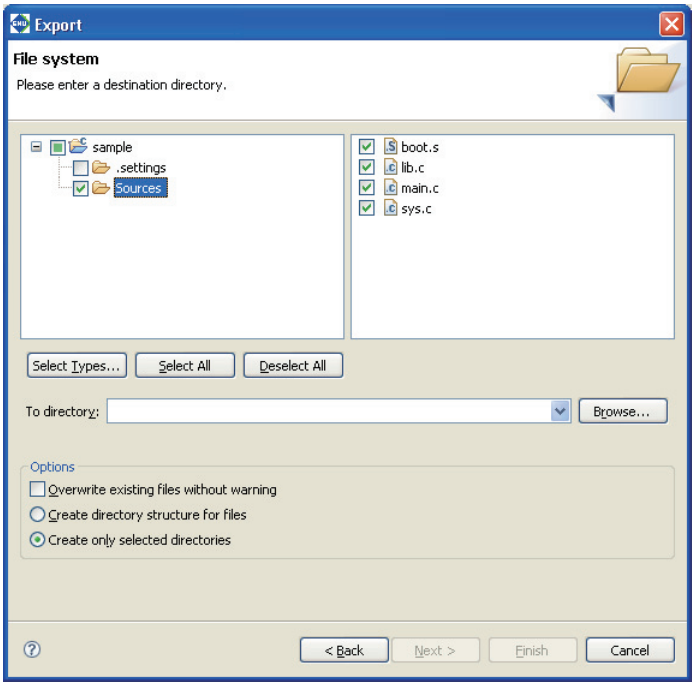
General
Team
< Back
Next >
Finish
Cancel

(3) Expand [General].



General
Archive File
Breakpoints
File System
Preferences
Team

(4) Select [File System] from the wizard list and click [Next>].


Export
File system
Please enter a destination directory.
sample
.settings
Sources
boot.s
lib.c
main.c
sys.c
Select Types...
Select All
Deselect All
To directory:
Browse...
Options
Overwrite existing files without warning
Create directory structure for files
Create only selected directories
< Back
Next >
Finish
Cancel

(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 5.10.6 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:

Notes:
- 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.
- Deleting a link to an external file or folder deletes only the link. The actual file or folder will not be deleted.

(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.

Linking to a file located outside the project folder

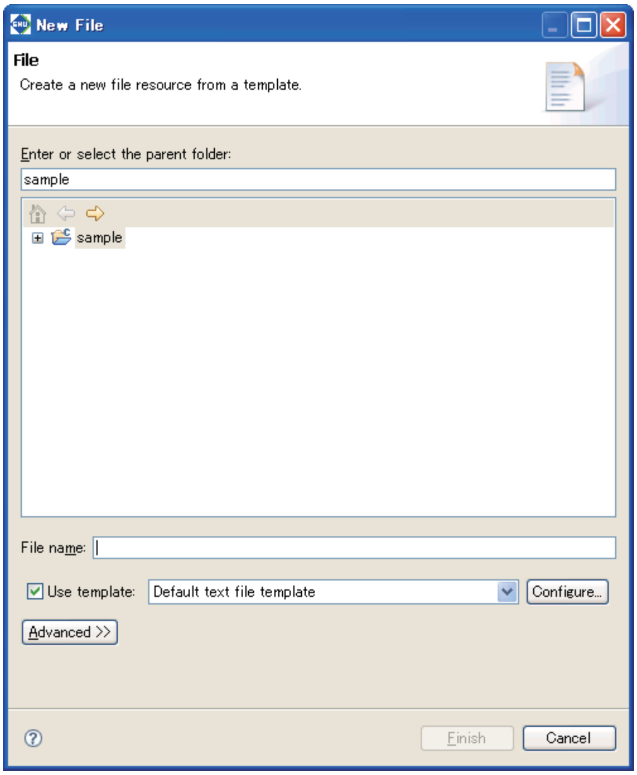
In team development, it is sometimes useful to reference source files located outside the project folder and include them in the build process.

A link to a file located outside the project folder can be established as follows:

Linked files can be edited just like those in the project folder.

(1) Do one of the following:
- Select [New] > [File from Template] from the [File] menu.
- Select [New] > [File from Template] from the context menu for the [C/C++ Projects] or [Navigator] view.
- Select [File from Template] from the [New] shortcut in the toolbar.
- Select [File from Template] from the [New C/C++ Source File] shortcut in the toolbar.

The [New File] dialog box is displayed.



New File

File

Create a new file resource from a template.

Enter or select the parent folder: sample

File name:

☑ Use template: Default text file template

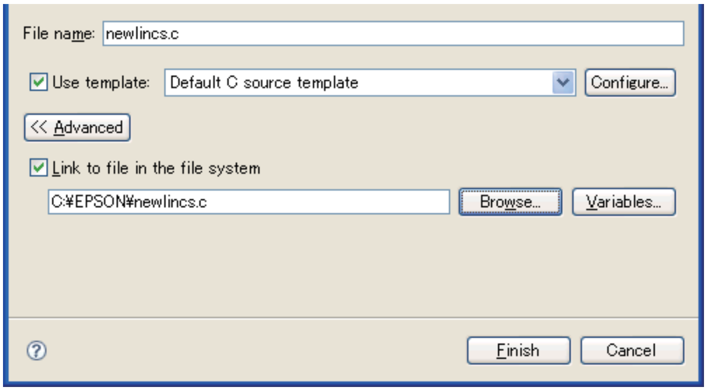
(2) In the tree view of folders, select the project or parent folder in which a new file is to be created.

(3) In [File name:], enter a name for the file to be created.

(4) Click the [Advanced>>] button.

(5) Select the [Link to file in the file system] checkbox.

(6) Click the [Browse] button and select a file to reference.



File name: newlincs.c

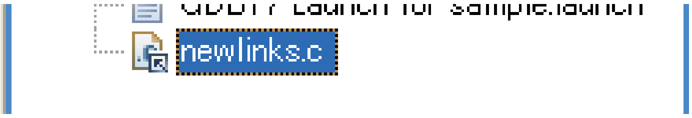
☑ Use template: Default C source template

☑ Link to file in the file system

C:¥EPSON¥newlincs.c

(7) Click the [Finish] button. Click the [Cancel] button to abort the process.

The newly-created file, appearing in tree view, can be opened by the editor.



Notes:
- Make sure the link source file name (File name) is the same as the link destination file name (Link to file in the file system).
- Be sure to select a source folder as the destination folder. (The project folder is a source folder by default.) If the file is not created in a source folder, it cannot be used in a build process.
- Use of the [Variables] button for path designation is not supported. Do not use the [Variables] button.

About object files generated from a link source

The object files for linked source files located outside the project folder will be generated directly under the project folder.

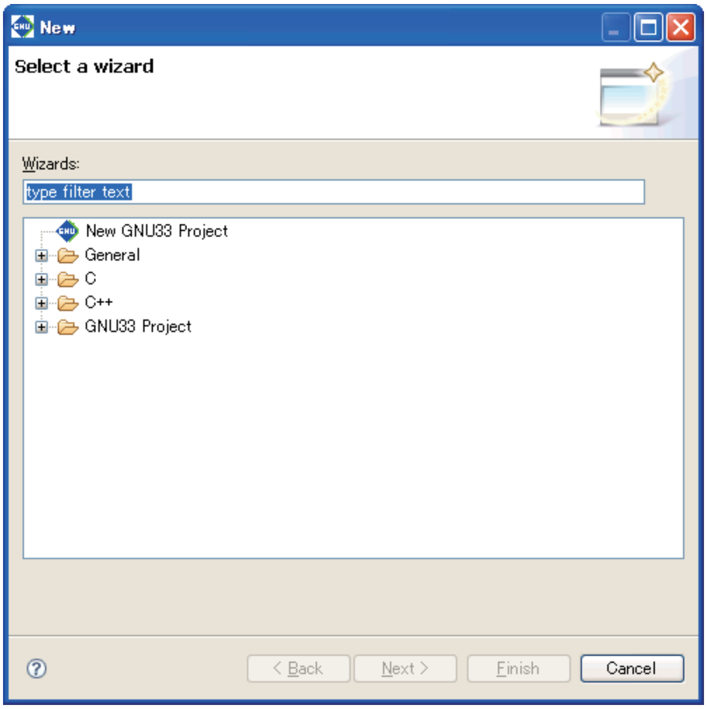
Linking to a folder located outside the project folder

In team development, it is sometimes useful to reference a group of source files in a folder located outside the project folder and include them in the build process.

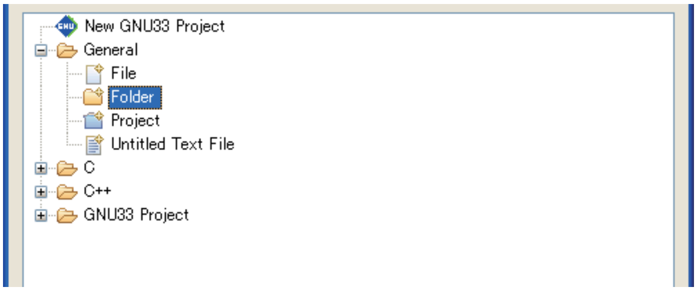
A link to a file located outside the project folder can be established as follows:

(1) Do one of the following:
- 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.

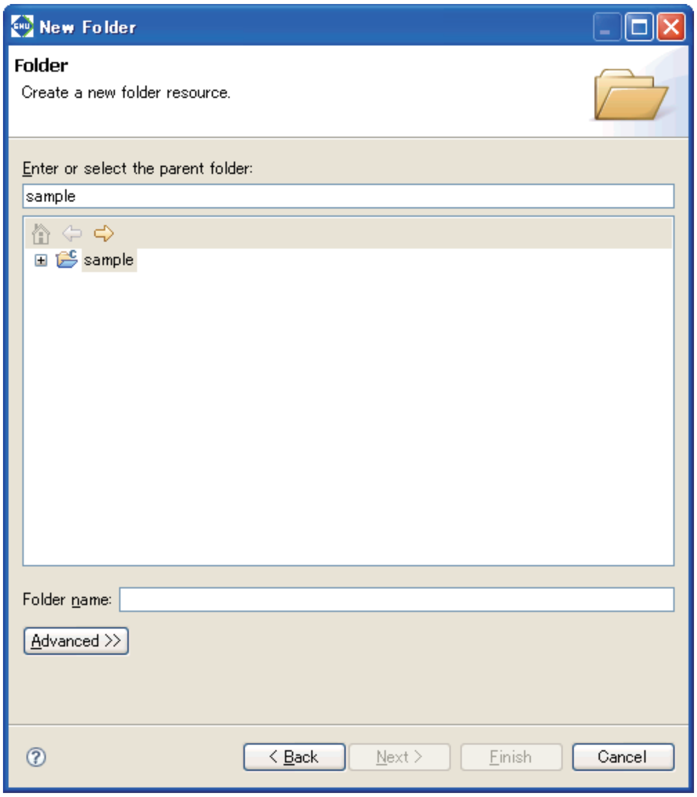
The [New] dialog box is displayed.


New
Select a wizard
Wizards:
type filter text
New GNU33 Project
General
C
C++
GNU33 Project
< Back
Next >
Finish
Cancel

(2) Select [Folder] from [General] in the list.


New GNU33 Project
General
File
Folder
Project
Untitled Text File
C
C++
GNU33 Project

(3) Click [Next >].


New Folder
Folder
sample
Advanced >>
< Back
Next >
Finish
Cancel

Create a new folder resource.

Enter or select the parent folder: sample

Folder name:

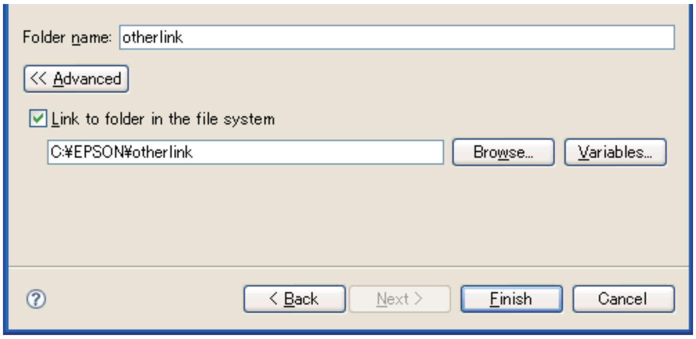
(4) In the tree view of folders, select the project or parent folder in which a new file is to be created.

(5) In [Folder name:], enter a name for the folder to be created.

(6) Click the [Advanced>>] button.

(7) Select the [Link to folder in the file system] checkbox.

(8) Click the [Browse] button and select a folder to reference.


<< Advanced
Browse...
Variables...
< Back
Next >
Finish
Cancel

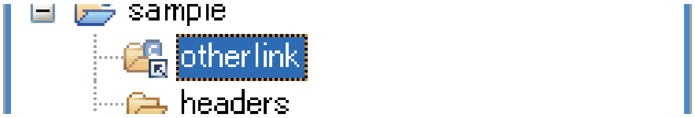
Folder name: otherlink

☑ Link to folder in the file system

C:¥EPSON¥otherlink

(9) Click the [Finish] button. Click the [Cancel] button to abort the process.
The newly-created folder will appear in tree view.

(10) Register the folder as a source folder. After registration as a source folder, a “C” mark will appear on the icon.


sample
otherlink
headers

Notes:
- Make sure the link source folder name (Folder name) is the same as the link destination folder name (Link to folder in the file system).
- Use of the [Variables] button for path designation is not supported. Do not use the [Variables] button.
- In order to build the source files that are in the linked folder, please specify the folder as a "source folder." For detailed information, refer to "Specifying a source folder."

About object files generated from a link source

The object files for linked source files located outside the project folder will be generated directly under the project folder.

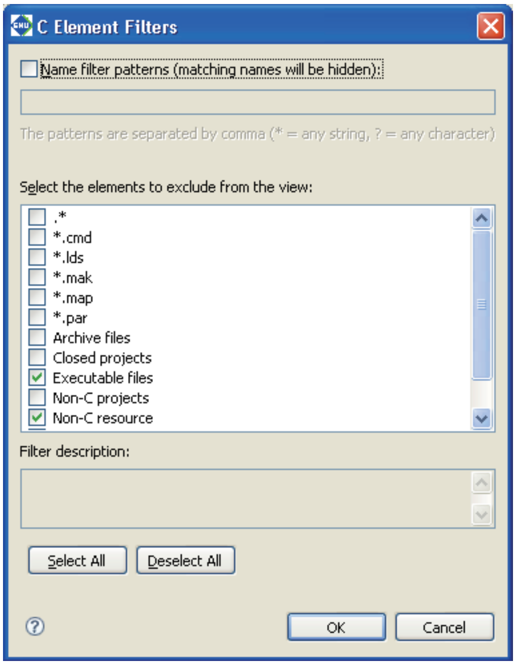
5.4.9 File Filter

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.



☐ Name filter patterns (matching names will be hidden):

The patterns are separated by comma (* = any string, ? = any character)

Select the elements to exclude from the view:

☐ .*
☐ *.cmd
☐ *.lds
☐ *.mak
☐ *.map
☐ *.par
☐ Archive files
☐ Closed projects
☑ Executable files
☐ Non-C projects
☑ Non-C resource

Filter description:

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.

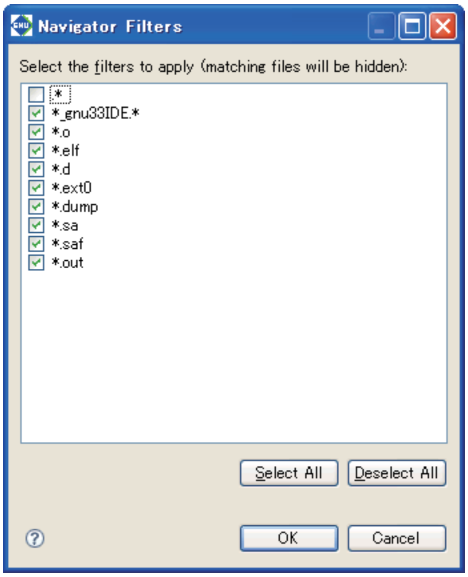
Select the [Name filter patterns:] check box and enter a character string in the text box. That allows you to filter out files whose names match the character string. You can use * (string) and ? (one character) wildcards.

The files listed below are not displayed in the initial settings for the [C/C++ Projects] view:

- Project files, etc. corresponding to ".*"
- Executable files (.elf files)
- Object files (.o files)
- Text files not associated with C/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 filters to apply (matching files will be hidden):

☐ *
☑ *_gnu33IDE.*
☑ *.o
☑ *.elf
☑ *.d
☑ *.ext0
☑ *.dump
☑ *.sa
☑ *.saf
☑ *.out

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 "*_gnu33IDE.*" (**IDE** files)
- Files corresponding to "*.ext0" (assembler source files output from the compiler)
- Files corresponding to "*.dump " (symbol files for 2pass build)
- Files corresponding to "*.sa" (S3 format files of elf executable files)
- Files corresponding to "*.saf" (S3 format files output by moto2ff)

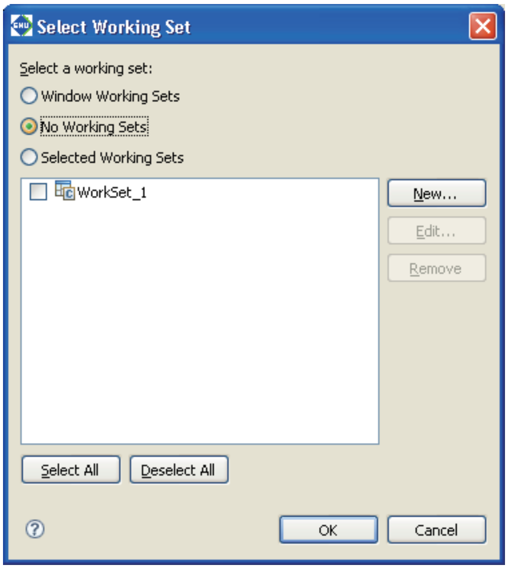
5.4.10 Working Set

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.


Select Working Set
Select a working set:
○ Window Working Sets
● No Working Sets
○ Selected Working Sets
☐ WorkSet_1
New...
Edit...
Remove
Select All
Deselect All
OK
Cancel

(2) Select a working set from the list and click [OK].

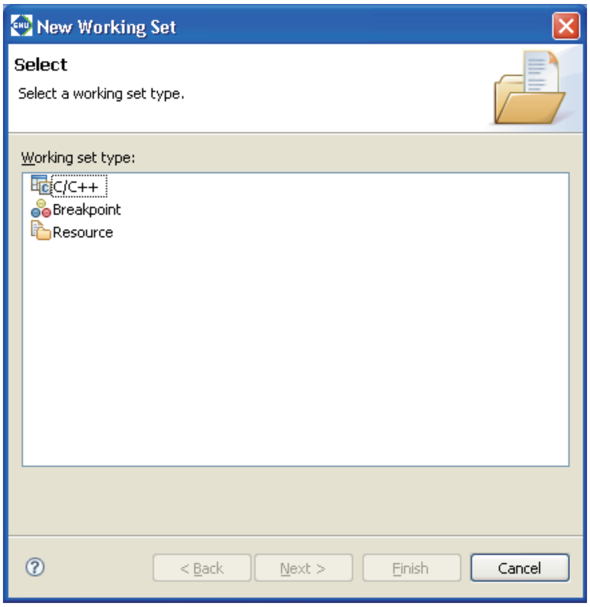
Only the resources defined in the selected working set will be displayed in the [C/C++ Projects] view.

To revert the display in the [C/C++ Projects] view, select [Deselect Working Set] from the toolbar menu (▽) in the [C/C++ Projects] view.

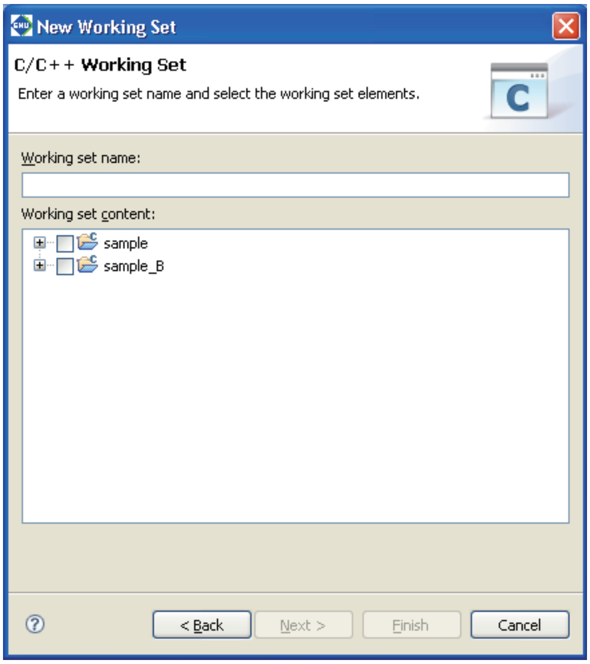
Creating a working set

(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.


New Working Set
Select
Select a working set type.
Working set type:
C/C++
Breakpoint
Resource
< Back
Next >
Finish
Cancel

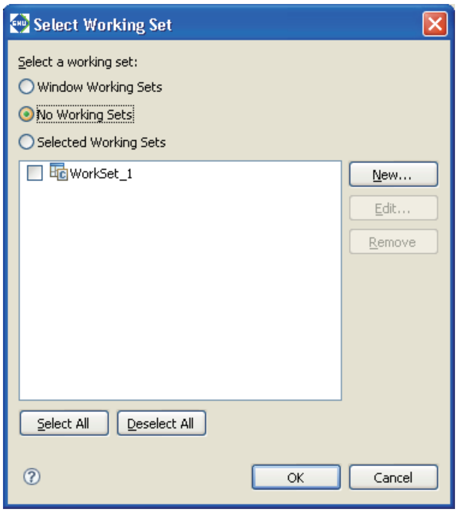
(3) Select [C/C++] from the [Working set type:] list and click [Next>].


New Working Set
C/C++ Working Set
Enter a working set name and select the working set elements.
Working set name:
Working set content:
sample
sample_B
< Back
Next >
Finish
Cancel

(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.


Select Working Set
Select a working set:
○ Window Working Sets
● No Working Sets
○ Selected Working Sets
☐ WorkSet_1
New...
Edit...
Remove
Select All
Deselect All
OK
Cancel

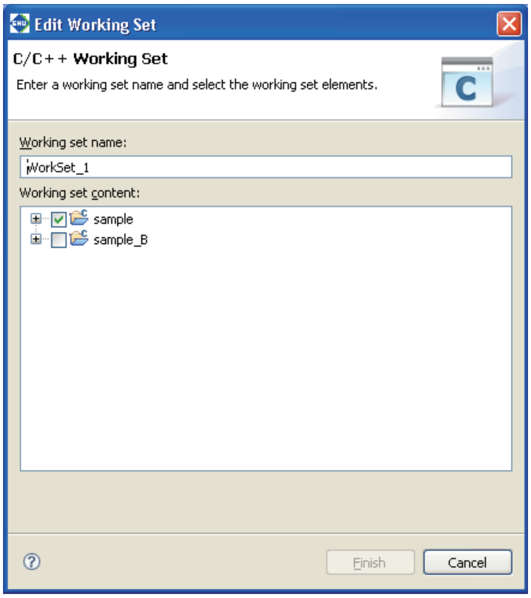
(7) Click [OK]. Your settings will be applied to the view. Clicking [Cancel] here will discard the settings.

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.


Edit Working Set
C/C++ Working Set
Enter a working set name and select the working set elements.
Working set content:
☑ sample
☐ sample_B
Finish
Cancel

Working set name: WorkSet_1

(3) Change the selected resources for the working set just as you did when creating a new working set. Click [Finish].

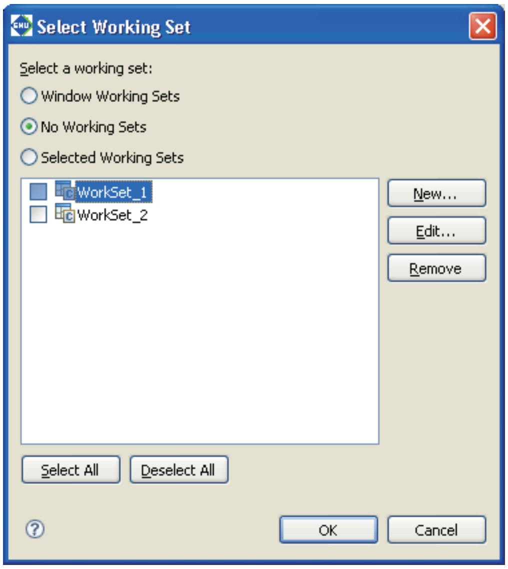
(4) Click [OK] to apply the settings to the view. Clicking [Cancel] here will discard these settings.

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.


Select Working Set
Select a working set:
○ Window Working Sets
● No Working Sets
○ Selected Working Sets
☐ WorkSet_1
☐ WorkSet_2
New...
Edit...
Remove
Select All
Deselect All
OK
Cancel

(2) Select a working set to delete from the list and click [Remove].
No working sets are selected in the display in this view.

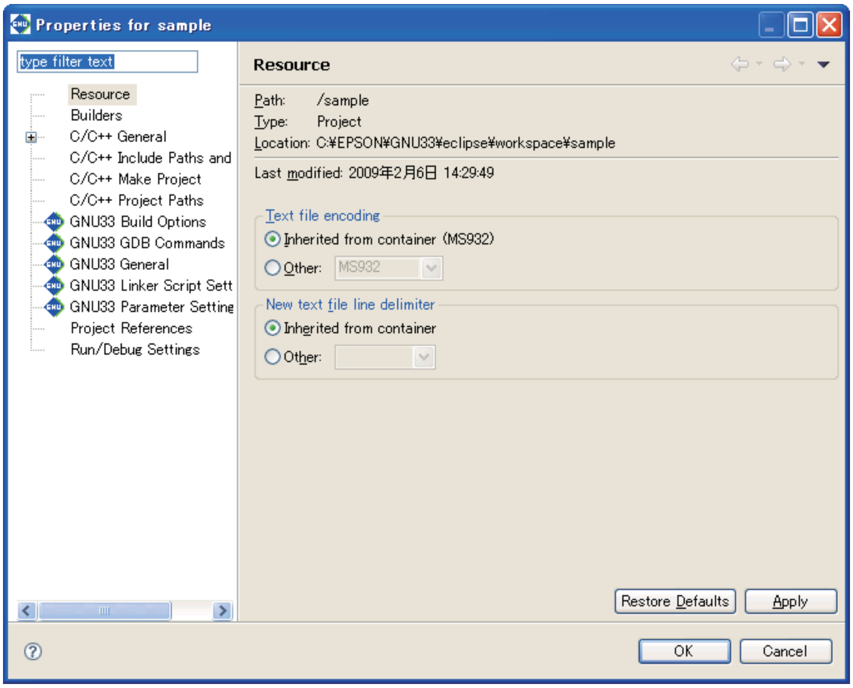
(3) Click the [OK] button.
To use another working set, select one from the list before clicking [OK].

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.


Properties for sample
type filter text
Resource
Builders
C/C++ General
C/C++ Include Paths and
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Commands
GNU33 General
GNU33 Linker Script Sett
GNU33 Parameter Setting
Project References
Run/Debug Settings
Resource
Restore Defaults
Apply
OK
Cancel

Path: /sample
Type: Project
Location: C:¥EPSON¥GNU33¥eclipse¥workspace¥sample

Last modified: 2009年2月6日 14:29:49

Text file encoding
◉ Inherited from container (MS932)
○ Other: MS932

New text file line delimiter
◉ Inherited from container
○ Other:

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. Resource

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++ General

General settings for C/C++.
Since the following items are not used by IDE, they can generally be disregarded in normal use.
For detailed information, select [C/C++ Development User Guide] > [Reference] > [C/C++ Properties] > [C/C++ Project Properties] > [C/C++ General] from the Help menu.

Code Style

Formatter settings.

Documentation

Selects the Help document to be used for a project.
Disregard in normal use.

File Types

Defines the names and extensions of files used as C resources.
Disregard in normal use.

Indexer

Makes settings for the indexer used in the C search and content assist.
Disregard in normal use.

Language Mappings

Changes the language and file mapping.
Disregard in normal use.

4. **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. **C/C++ Make Project**
   Set make conditions.

6. **C/C++ Project Paths**
   Set the destination, etc. to which the source directory and generated files are to be output.

7. **GNU33 Build Options**
   Set the command line options for the compiler, assembler, and linker.

8. **GNU33 GDB Commands**
   Edit a debugger startup command file.

9. **GNU33 General**
   Select the target processor.

10. **GNU33 Linker Script Settings**
    Edit the linker script.

11. **GNU33 Parameter Settings**
    Edit the parameter file used in the debugger.

12. **Project references**
    Select other projects to be referenced by the current project.

13. **Refactoring History**
    Displays the refactoring history.

14. **Run/Debug Settings**
    A launch configuration can be specified for the file being selected.
    Disregard in normal use.

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/C++ editor

Used to create and edit C/C++ sources (*.c, *.cpp, *.cxx, *.cc, *.C) and header files (*.h, *.hh, *.hpp, *.hxx, *.H). This editor highlights C/C++ reserved words, comments, and strings. The editor provides a "content assist" feature that allows you to enter C/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.

.c, .h

2. Assembler editor

Used to create and edit assembler sources (*.s, *.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.

.S

The [C/C++ Projects] view cannot display the symbols and labels in assembly source files in its tree list.

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/C++ editor or assembler editor (i.e., launch the C/C++ or assembler editor by default when opened) are registered as C/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.

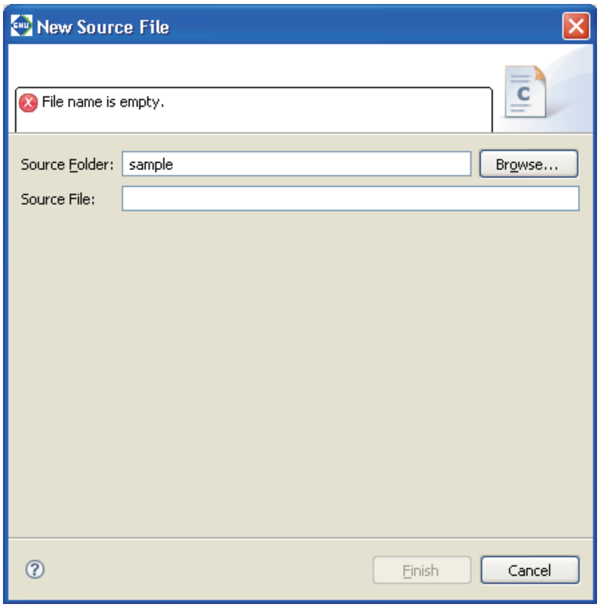
Double clicking on a "*.cmd" or "*.bat" file launches Command Prompt. To edit a file of one of these types, drag and drop it on the editor area.

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.



File name is empty.

Source Folder: sample

Source File:

(2) Enter the name of the file you want to create in the [Source File:] text box. To create a C/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 from Template]	[New]>[Source File],[Header File],[File from Template] (C/C++ Projects, Navigator)	📄▾, C▾
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)	–	–

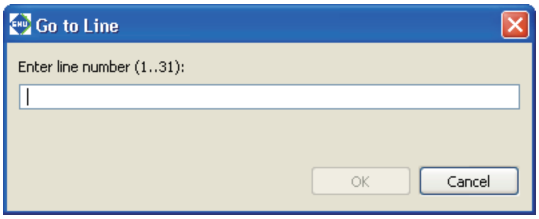
5.5.3 Editing Functions for C/C++ Source Files

In addition to the basic editing functions shown above, the C/C++ editor provides features specific to C/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.

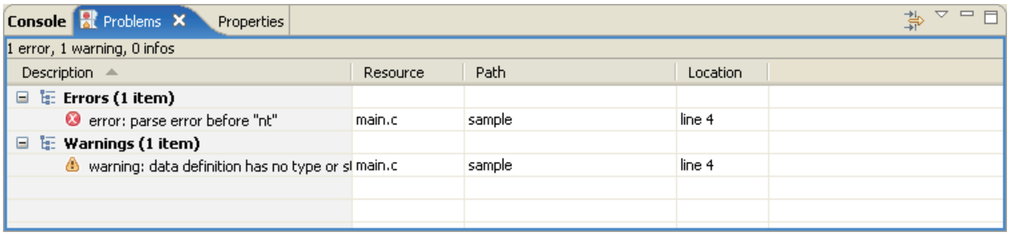


(2) Enter a line number in the text box and click [OK].
Control will jump to the specified line.

With its default settings, the C/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

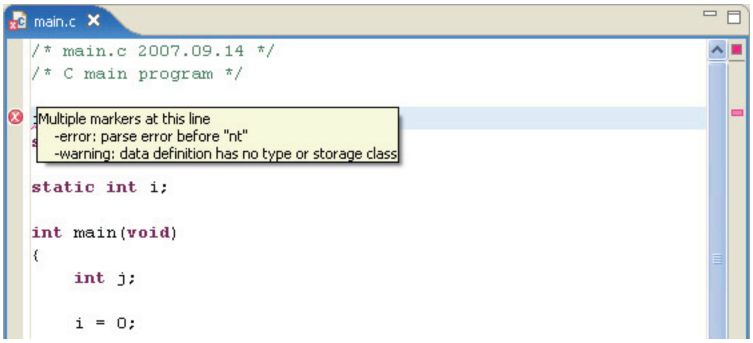
When an error occurs during a build process, a list of errors and their locations are displayed in the [Problems] view.



1 error, 1 warning, 0 infos

Description	Resource	Path	Location
Errors (1 item)			
error: parse error before "nt"	main.c	sample	line 4
Warnings (1 item)			
warning: data definition has no type or st	main.c	sample	line 4

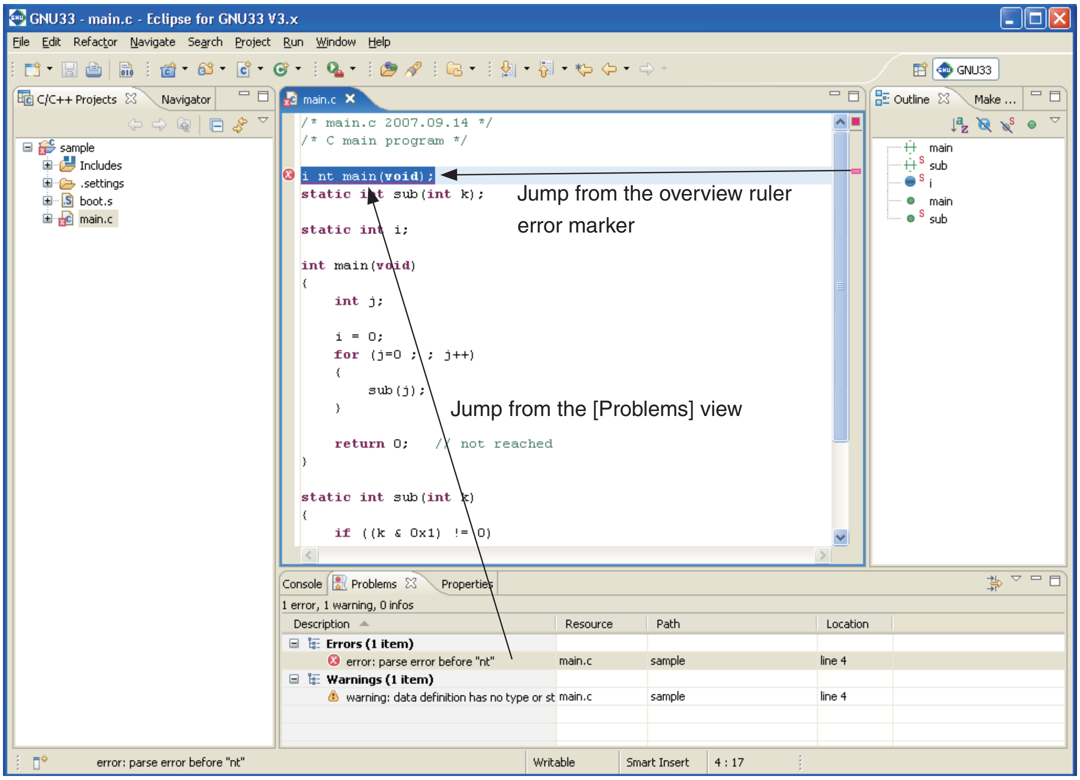
If an error occurs in the source file currently open in the editor, the line with the error is marked with a symbol (⊗). Hover the mouse cursor over this marker to see the nature of the error.



```
/* main.c 2007.09.14 */
/* C main program */
```

Multiple markers at this line
-error: parse error before "nt"
-warning: data definition has no type or storage class

```
static int i;

int main(void)
{
    int j;

    i = 0;
```



```
/* main.c 2007.09.14 */
/* C main program */

i nt main(void);
static int sub(int k);

static int i;

int main(void)
{
    int j;

    i = 0;
    for (j=0 ; ; j++)
    {
        sub(j);
    }

    return 0;    // not reached
}

static int sub(int k)
{
    if ((k & 0x1) != 0)
```

Jump from the overview ruler error marker

Jump from the [Problems] view

1 error, 1 warning, 0 infos

Description	Resource	Path	Location
Errors (1 item)			
error: parse error before "nt"	main.c	sample	line 4
Warnings (1 item)			
warning: data definition has no type or st	main.c	sample	line 4

error: parse error before "nt" | Writable | Smart Insert | 4 : 17

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/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.

Launching external editor by specifying line number

If an external editor has been set according to the procedure described in Section 5.5.10, "Launching External Editor by Specifying Line Number", you can jump to the error-generating line by using the context menu.

For details, refer to Section 5.5.10, "Launching External Editor by Specifying Line Number".

Incremental search

You can use the incremental search function in the C/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 2007.09.14 *
/* C main program */
          :
```

[m] key entered

```
          :
/* main.c 2007.09.14 *
/* C main program */
          :
```

[a] [i] [n] key entered

```
          :
/* main.c 2007.09.14 *
/* 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 2007.09.14 *
/* C main program */
          :
```

[↓] key entered

```
          :
/* main.c 2007.09.14 *
/* 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/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/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 "/*".

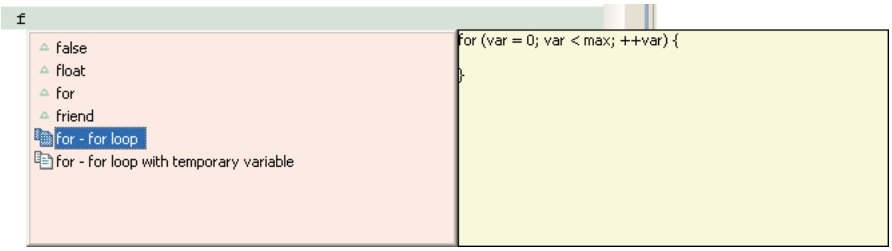
Content assist

The C/C++ editor has a content assist facility that allows the user to select a C/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.



```
for (var = 0; var < max; ++var) {

}
```

(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

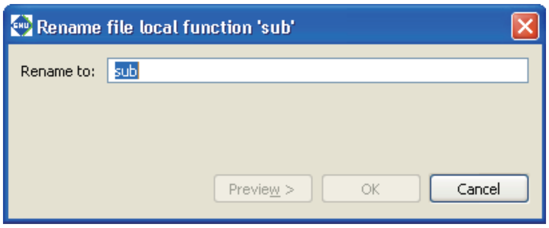
Note: The refactoring is an Eclipse CDT function under development. After this function is used to edit source files, make sure that the program can run without a problem.

You can change the names of variables, class, 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.

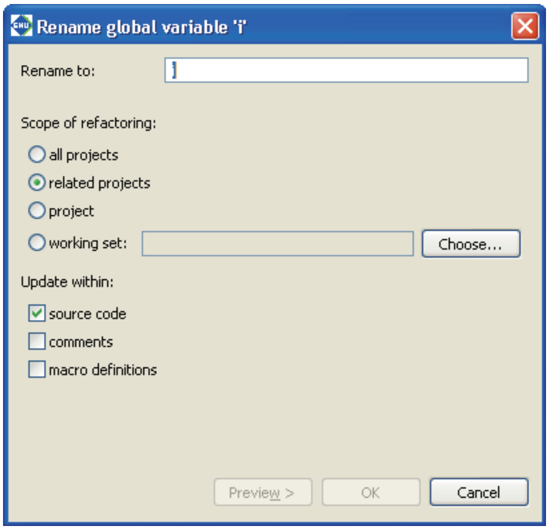
For file scope variables/functions (static variables, static functions, local variables, function parameters, etc.)



Rename file local function 'sub'

Rename to: sub

For other global variables/functions (extern variables, extern functions, etc.)



Rename global variable 'i'

Rename to: i

Scope of refactoring:
○ all projects
● related projects
○ project
○ working set:

Update within:
☑ source code
☐ comments
☐ macro definitions

(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. (global variables/functions)

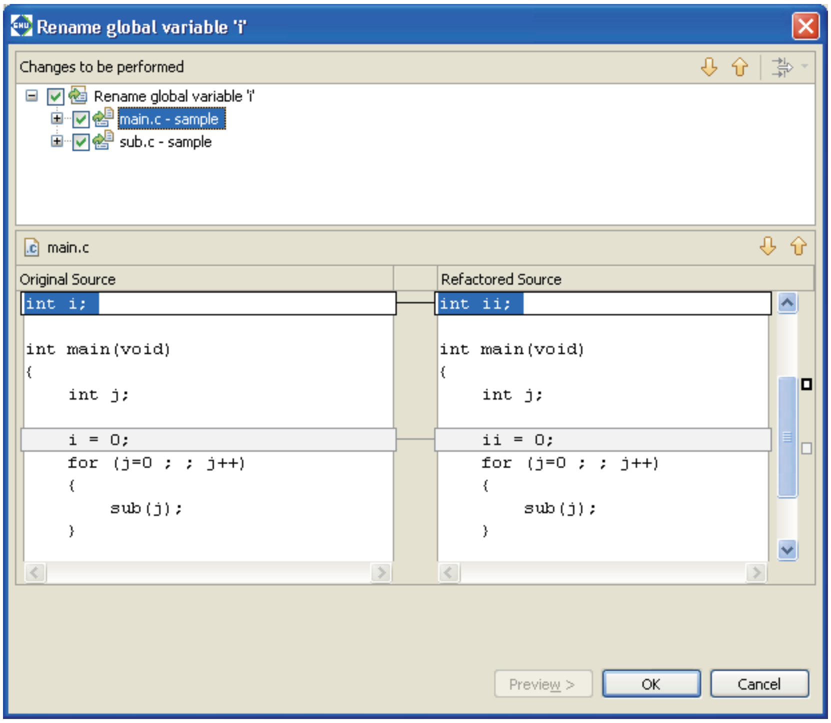
[all projects]	All the opened projects will be in the scope of renaming.
[related projects]	All the projects related to the project being currently edited will be in the scope of renaming.
[project]	The project being currently edited only will be in the scope of renaming.
[working set:]	Projects or files 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. (global variables/functions)

[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.

When renaming a file scope variable/function, the effective range is fixed at the file currently edited.

(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.


Rename global variable 'i'
Changes to be performed
☑ Rename global variable 'i'
☑ main.c - sample
☑ sub.c - sample
main.c
Original Source
Refactored Source
Preview >
OK
Cancel

```
int i;

int main(void)
{
    int j;

    i = 0;
    for (j=0 ; ; j++)
    {
        sub(j);
    }
```

```
int ii;

int main(void)
{
    int j;

    ii = 0;
    for (j=0 ; ; j++)
    {
        sub(j);
    }
```

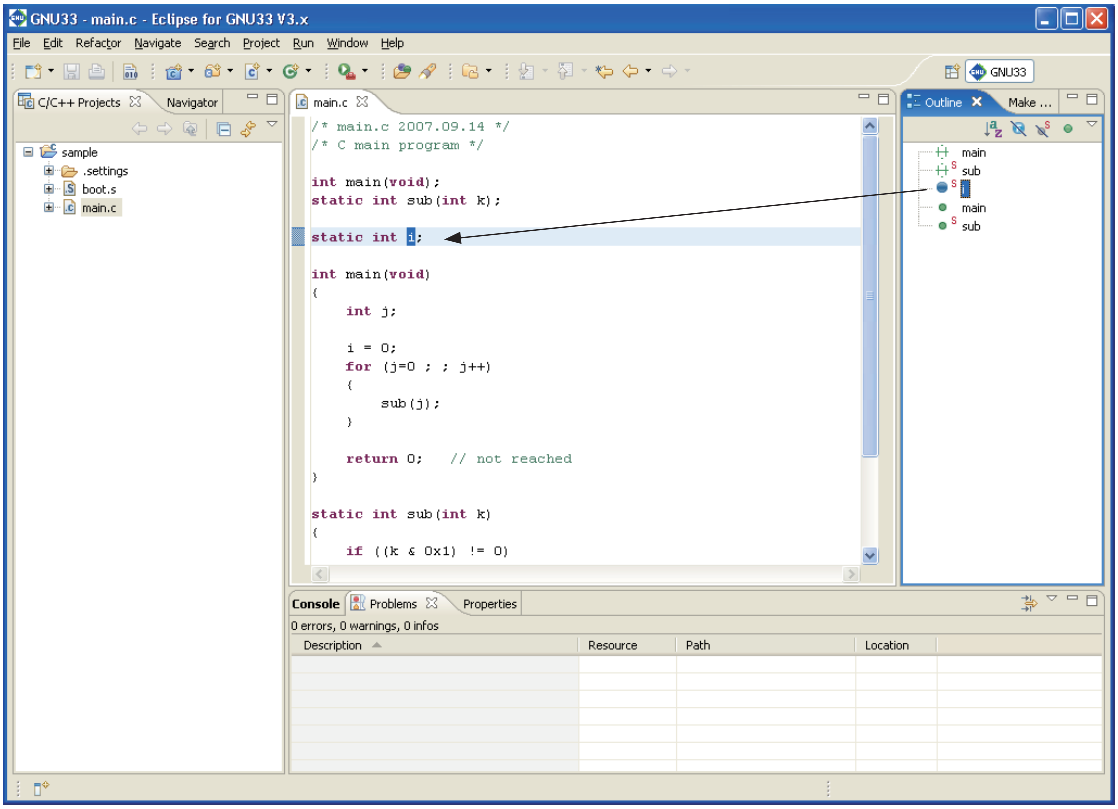
Use the arrow buttons to check the locations of the elements that will be renamed.

(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/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.


GNU33 - main.c - Eclipse for GNU33 V3.x
File Edit Refactor Navigate Search Project Run Window Help
GNU33
C/C++ Projects
Navigator
sample
.settings
boot.s
main.c
Outline
Make ...
main
sub
i
main
sub
Console
Problems
Properties
0 errors, 0 warnings, 0 infos
Description
Resource
Path
Location

```
/* main.c 2007.09.14 */
/* C main program */

int main(void);
static int sub(int k);

static int i;

int main(void)
{
    int j;

    i = 0;
    for (j=0 ; ; j++)
    {
        sub(j);
    }

    return 0;    // not reached
}

static int sub(int k)
{
    if ((k & 0x1) != 0)
```

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.

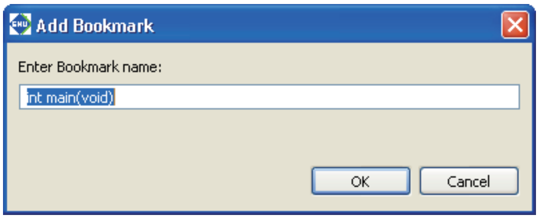
Attaching a bookmark

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...].

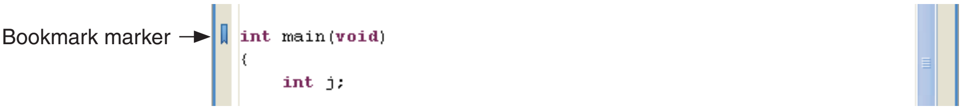
This displays the [Add Bookmark] dialog box.



Enter Bookmark name: int main(void)

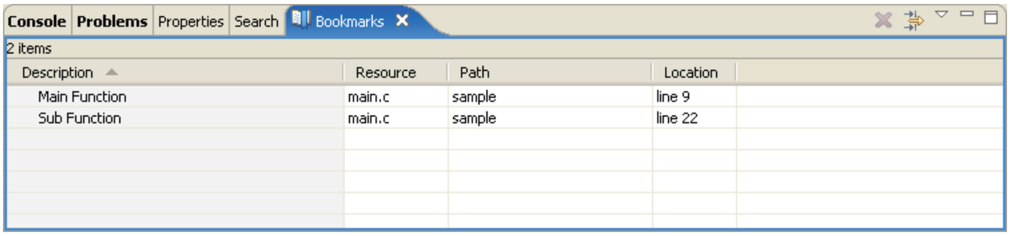
(3) Set a bookmark name. Use the name displayed in the [Enter Bookmark name:] text box unchanged, or enter another name and click [OK].

A bookmark marker appears in the editor's marker bar.



```
int main(void)
{
    int j;
```

Open the [Bookmarks] view. The bookmark just set has been added to the list.



2 items

Description	Resource	Path	Location
Main Function	main.c	sample	line 9
Sub Function	main.c	sample	line 22

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].

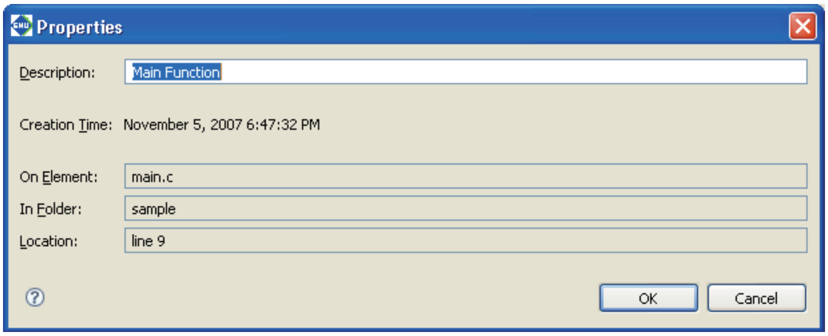
Removing in the [Bookmarks] view

(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.



Description: Main Function

Creation Time: November 5, 2007 6:47:32 PM

On Element: main.c

In Folder: sample

Location: line 9

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.

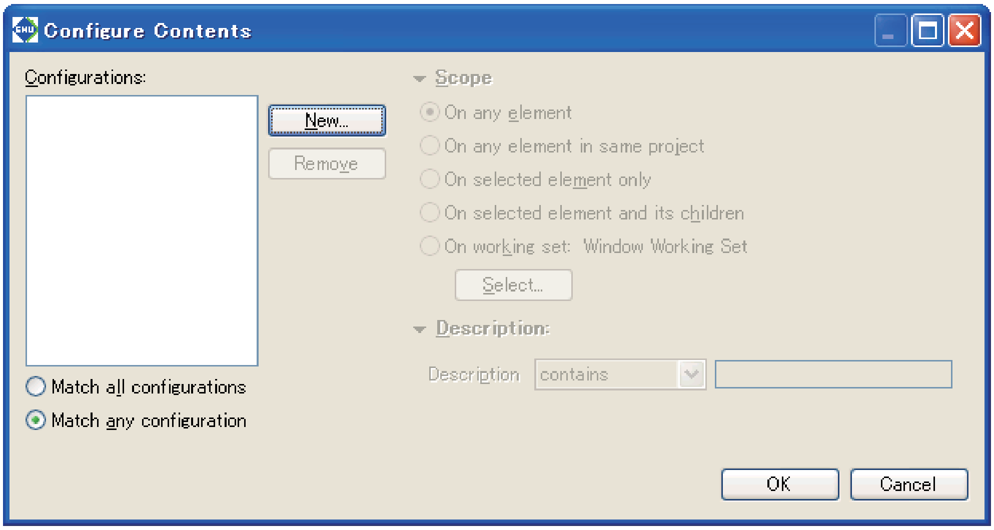
Filters

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.

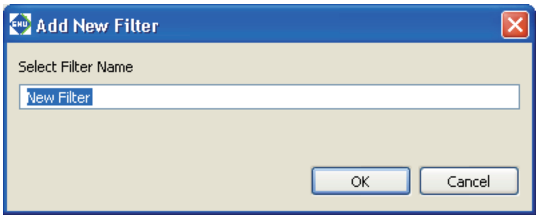
(1) Activate [Bookmarks] view.

(2) Select [Configure Contents...] from the View menu (▽).

This displays the [Configure Contents] dialog box.


Configure Contents
Configurations:
New...
Remove
Scope
On any element
On any element in same project
On selected element only
On selected element and its children
On working set: Window Working Set
Select...
Description:
Description
contains
Match all configurations
Match any configuration
OK
Cancel

(3) Click the [New] button.

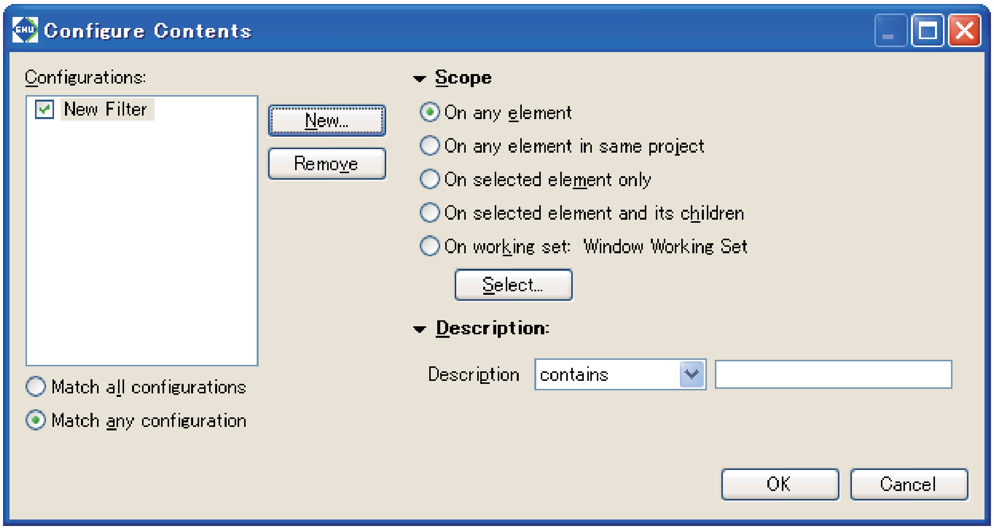

Add New Filter
Select Filter Name
New Filter
OK
Cancel

Enter the name of the filter to be configured and click [OK].

(4) If other filters are shown in [Configurations:], deselect the checkboxes for those filters and select the checkbox for the newly-created filter.

(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.



Configurations:
☑ New Filter

○ Match all configurations
● Match any configuration

Scope
● On any element
○ On any element in same project
○ On selected element only
○ On selected element and its children
○ On working set: Window Working Set

Description:
Description: contains

(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 [Configurations:] list. Or select them from the submenu of the [Show] view menu.

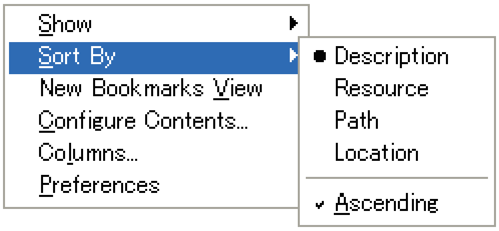
Refer to Section 5.10.7 for information on settings made in the [Configure Contents] dialog box.

Sorting

You can prioritize items and sort the displayed bookmarks in order of prioritized items.

(1) Activate the [Bookmarks] view.

(2) Select [Sort by] from the View menu (▽) and select the items in the list you wish to prioritize over others when sorting the list. Selecting [Ascending] sorts and arranges items in ascending order. Deselecting [Ascending] sorts and arranges items in descending order.


Show
Sort By
New Bookmarks View
Configure Contents...
Columns...
Preferences
● Description
Resource
Path
Location
✓ Ascending

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

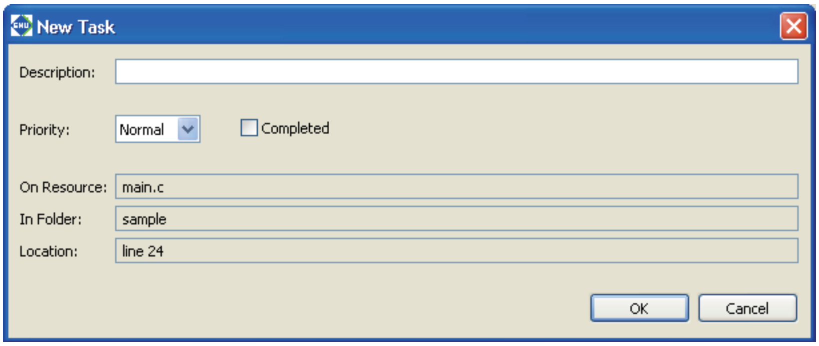
Do the following to create a task:

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.



New Task

Description:

Priority: Normal ☐ Completed

On Resource: main.c

In Folder: sample

Location: line 24

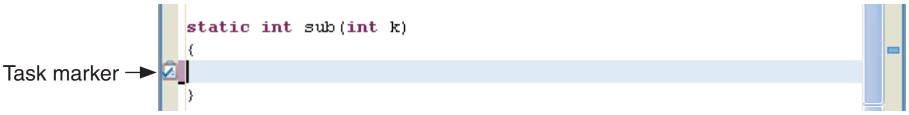
(3) Enter a task description in the [Description:] text box.

(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.

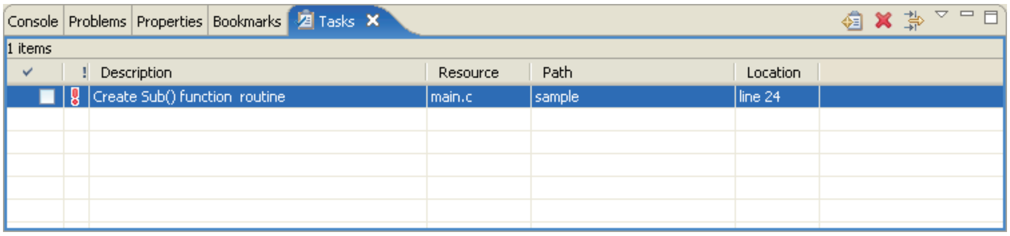
(6) Click the [OK] button.

A task marker is displayed in the editor's marker bar.



```
static int sub(int k)
{

}
```

Open the [Tasks] view. The task created should appear in the list.



1 items

✓	!	Description	Resource	Path	Location
☐	!	Create Sub() function routine	main.c	sample	line 24

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.

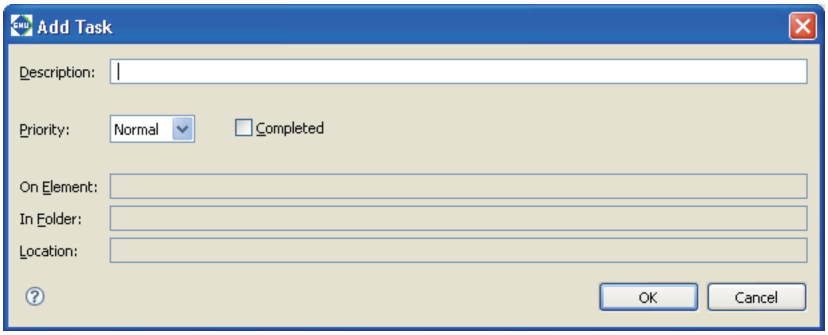
Click in the [Description] column to change a task description.

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.

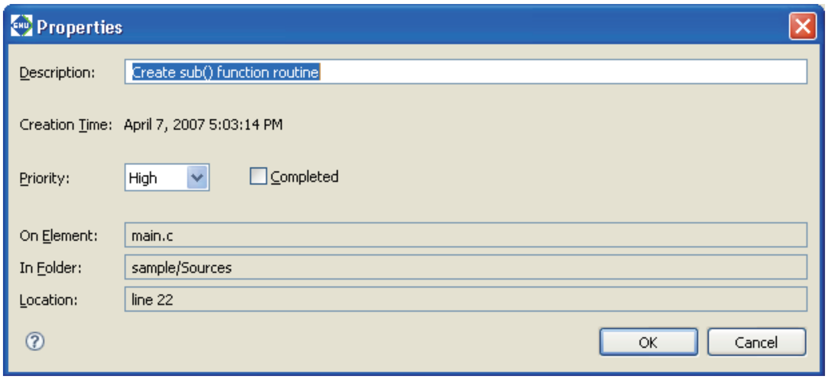
(1) Select [Delete Completed Tasks] from the context menu of the [Tasks] view.

(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.



Description: Create sub() function routine

Creation Time: April 7, 2007 5:03:14 PM

Priority: High

☐ Completed

On Element: main.c

In Folder: sample/Sources

Location: line 22

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.

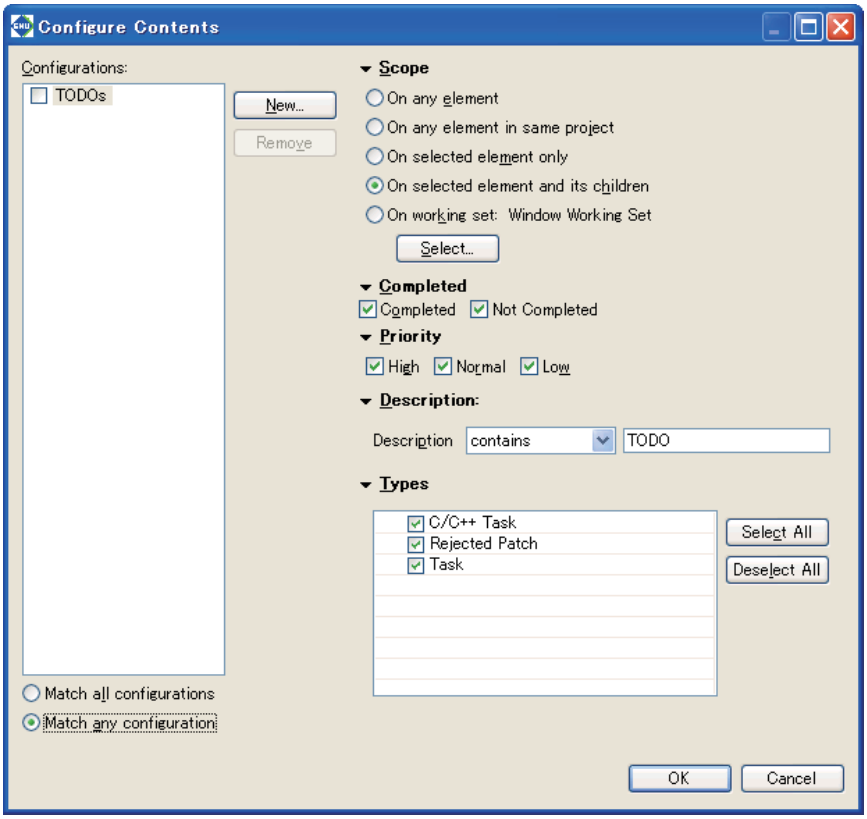
Filters

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) Select [Configure Contents...] from the View menu (▽).

This displays the [Configure Contents] dialog box.



Configurations:
☐ TODOs

Scope
○ On any element
○ On any element in same project
○ On selected element only
● On selected element and its children
○ On working set: Window Working Set

Completed
☑ Completed ☑ Not Completed

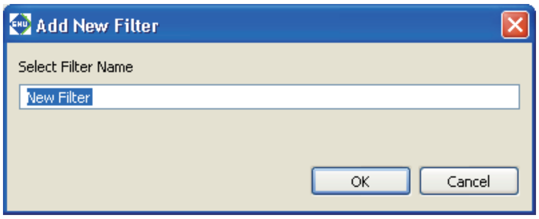
Priority
☑ High ☑ Normal ☑ Low

Description:
Description contains TODO

Types
☑ C/C++ Task
☑ Rejected Patch
☑ Task

○ Match all configurations
● Match any configuration

(3) Click the [New] button.



Select Filter Name
New Filter

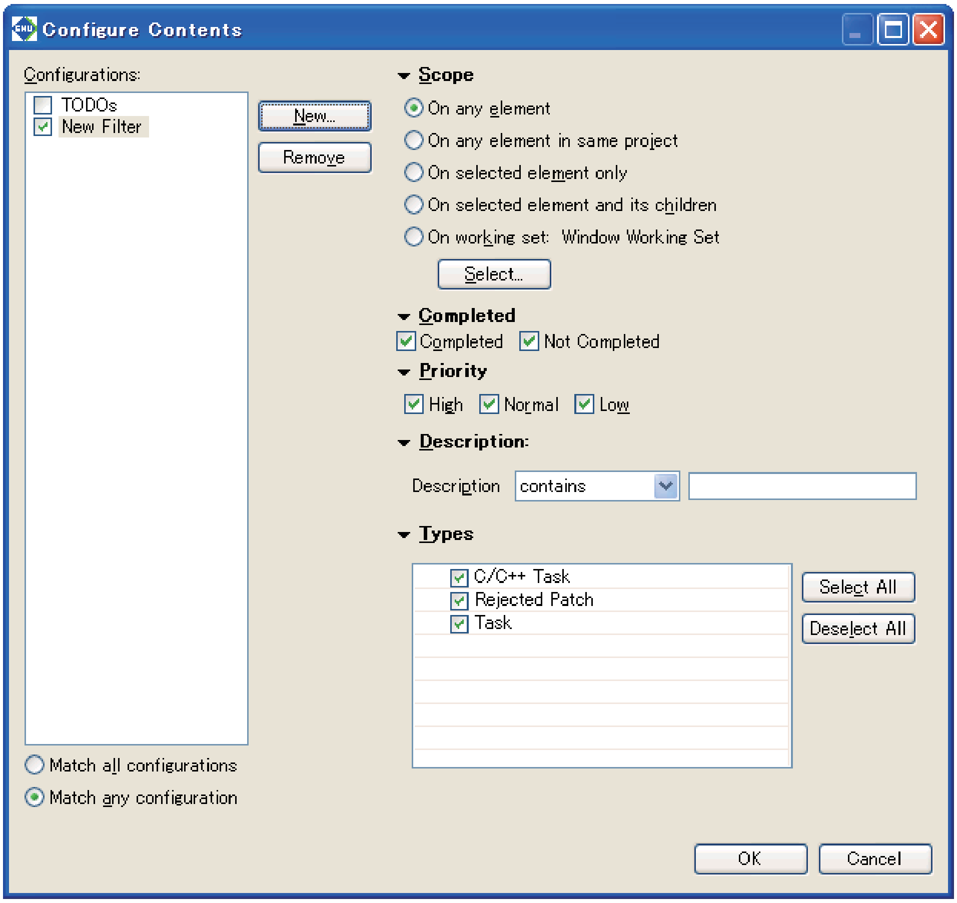
Enter the name of the filter to be configured and click [OK].

(4) If other filters are shown in [Configurations:], deselect the checkboxes for those filters and select the checkbox for the newly-created filter.

(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.
[Completed] - [Completed] / [Not Completed]	Select to display “Completed” or “Not completed.”
[Priority] - [High] / [Normal] / [Low]	Select to display tasks with a specific priority setting (High, Normal, Low).
[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.



Configurations:
☐ TODOs
☑ New Filter

○ Match all configurations
● Match any configuration

Scope
● On any element
○ On any element in same project
○ On selected element only
○ On selected element and its children
○ On working set: Window Working Set

Completed
☑ Completed ☑ Not Completed

Priority
☑ High ☑ Normal ☑ Low

Description:
Description contains

Types
☑ C/C++ Task
☑ Rejected Patch
☑ Task

(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 [Configurations:] list. Or select them from the submenu of the [Shows] view menu.

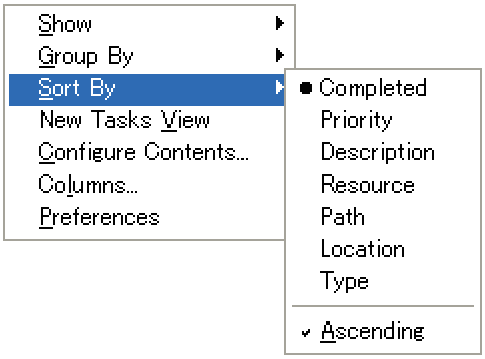
Refer to Section 5.10.7 for information on settings made in the [Configure Contents] dialog box.

Sorting

You can prioritize items and sort the displayed bookmarks in order of prioritized items.

(1) Activate the [Bookmarks] view.

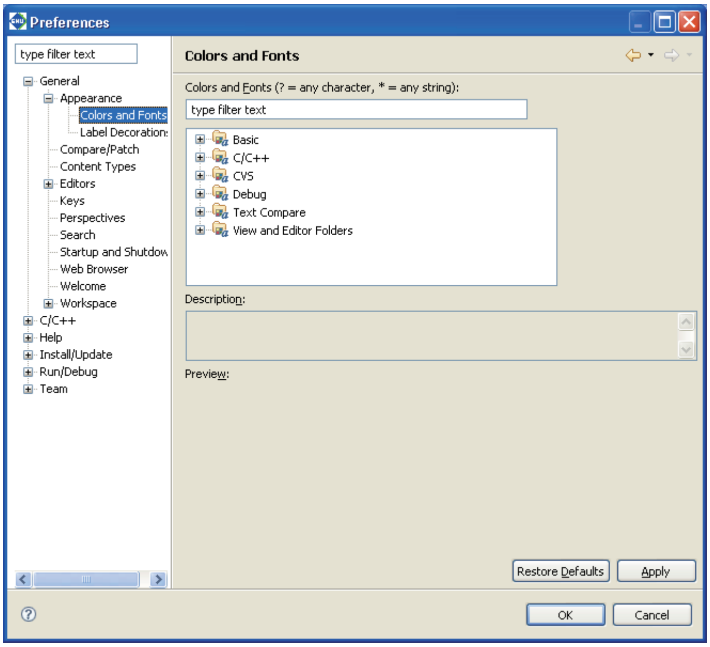
(2) Select [Sort by] from the View menu (▽) and select the items in the list you wish to prioritize over others when sorting the list. Selecting [Ascending] sorts and arranges items in ascending order. Deselecting [Ascending] sorts and arranges items in descending order.


Show
Group By
Sort By
New Tasks View
Configure Contents...
Columns...
Preferences
● Completed
Priority
Description
Resource
Path
Location
Type
✓ Ascending

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.

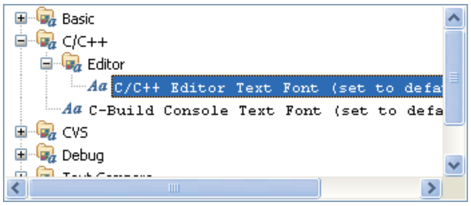
Text fonts and colors ([General] > [Appearance] > [Colors and Fonts])


Preferences
type filter text
General
Appearance
Colors and Fonts
Label Decoration:
Compare/Patch
Content Types
Editors
Keys
Perspectives
Search
Startup and Shutdow
Web Browser
Welcome
Workspace
C/C++
Help
Install/Update
Run/Debug
Team
Colors and Fonts
Colors and Fonts (? = any character, * = any string):
type filter text
Basic
C/C++
CVS
Debug
Text Compare
View and Editor Folders
Description:
Preview:
Restore Defaults
Apply
OK
Cancel

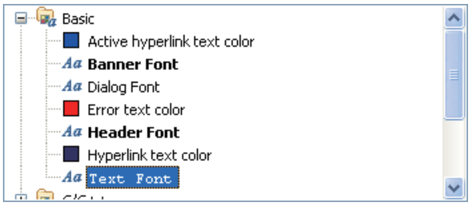
Here, you can change the default text fonts used by the C/C++ editor and assembler editor.

(1) To change fonts for the C/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/C++ editor fonts


Basic
C/C++
Editor
C/C++ Editor Text Font (set to defa
C-Build Console Text Font (set to defa
CVS
Debug

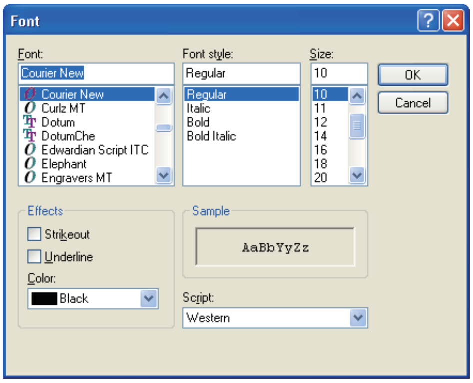
Assembler editor fonts


Basic
Active hyperlink text color
Banner Font
Dialog Font
Error text color
Header Font
Hyperlink text color
Text Font

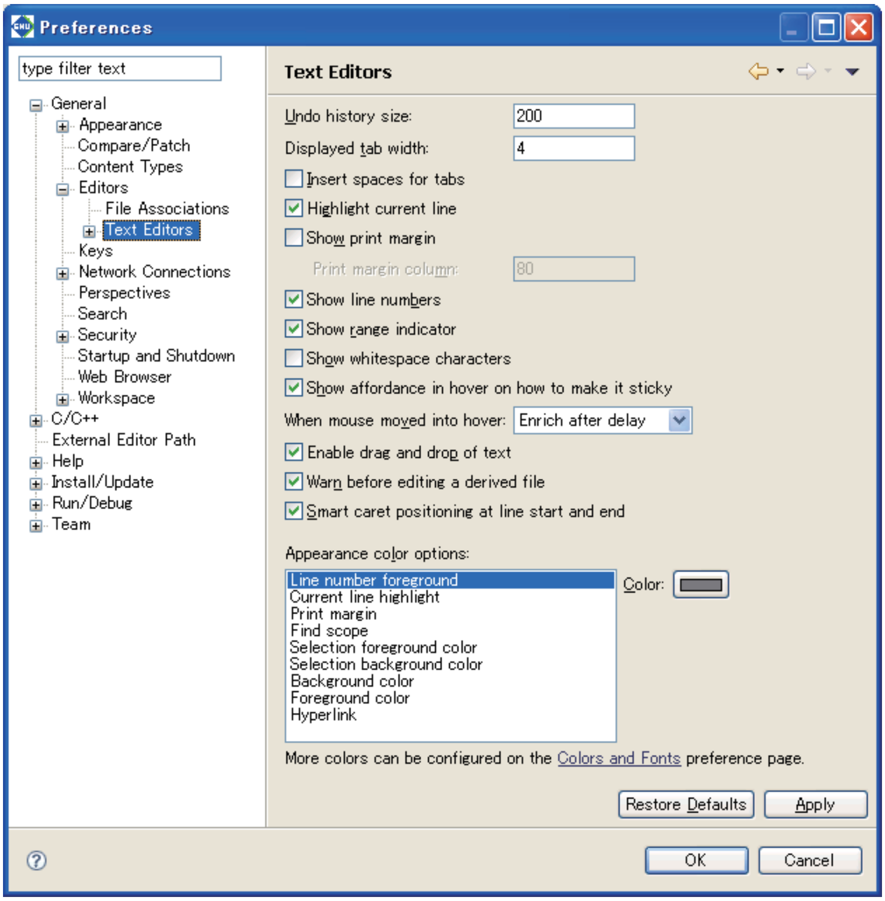
(2) Click [Change...] to display the font select dialog box. Select a font, font style, and display color in the dialog box.

Or use the [Use System Font] button to select the standard Windows font.

(3) Click [Apply] or [OK] to complete the settings.


Font
Font:
Courier New
Courier New
Curlz MT
Dotum
DotumChe
Edwardian Script ITC
Elephant
Engravers MT
Font style:
Regular
Regular
Italic
Bold
Bold Italic
Size:
10
10
11
12
14
16
18
20
OK
Cancel
Effects
Strikeout
Underline
Color:
Black
Sample
AaBbYyZz
Script:
Western

Changing the editor tab size and displaying line numbers ([General] > [Editors] > [Text Editors])


Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
File Associations
Text Editors
Keys
Network Connections
Perspectives
Search
Security
Startup and Shutdown
Web Browser
Workspace
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
Restore Defaults
Apply
OK
Cancel

Text Editors

Undo history size: 200

Displayed tab width: 4

☐ Insert spaces for tabs

☑ Highlight current line

☐ Show print margin

Print margin column: 80

☑ Show line numbers

☑ Show range indicator

☐ Show whitespace characters

☑ Show affordance in hover on how to make it sticky

When mouse moved into hover: Enrich after delay

☑ Enable drag and drop of text

☑ Warn before editing a derived file

☑ Smart caret positioning at line start and end

Appearance color options:

- Line number foreground
- Current line highlight
- Print margin
- Find scope
- Selection foreground color
- Selection background color
- Background color
- Foreground color
- Hyperlink

Color:

More colors can be configured on the Colors and Fonts preference page.

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.

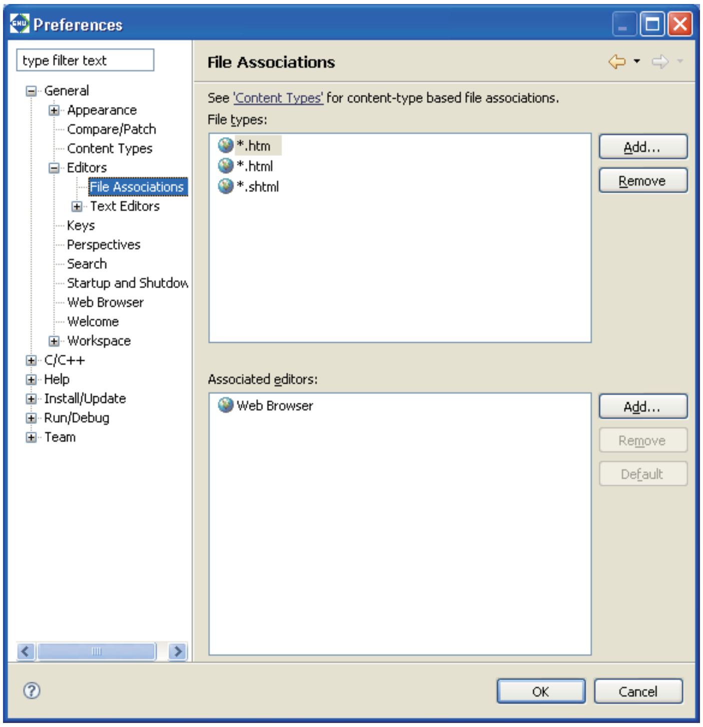
You can also set highlighting and other options on this page.

5.5.9 Using an External Editor

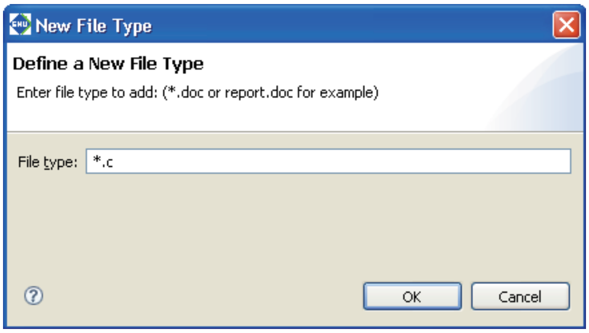
You can register a preferred editor to launch from the **IDE** for resource editing.
The procedure for registering an external editor is described below.

(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.

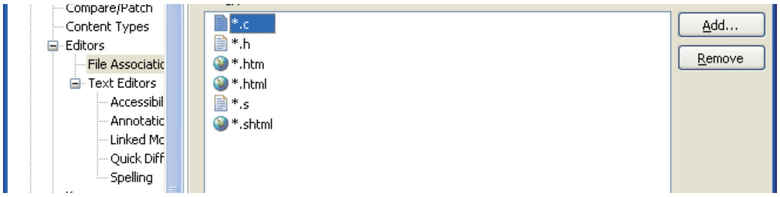

Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
File Associations
Text Editors
Keys
Perspectives
Search
Startup and Shutdow
Web Browser
Welcome
Workspace
C/C++
Help
Install/Update
Run/Debug
Team
File Associations
See 'Content Types' for content-type based file associations.
File types:
*.htm
*.html
*.shtml
Add...
Remove
Associated editors:
Web Browser
Add...
Remove
Default
OK
Cancel

(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, *.cc, *.cpp, *.cxx, *.h, *.hh, *.hpp, *.hxx, *.s) to add it to the list.

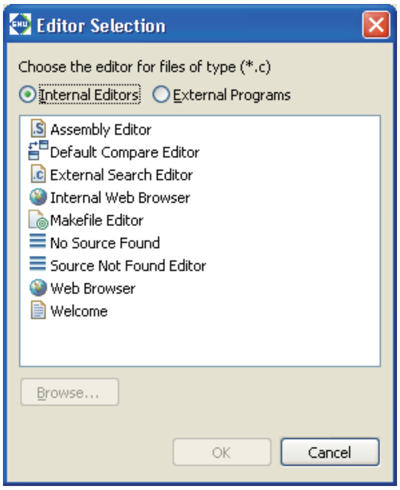

New File Type
Define a New File Type
OK
Cancel

Enter file type to add: (*.doc or report.doc for example)

File type: *.c


Compare/Patch
Content Types
Editors
File Associatic
Text Editors
Accessibil
Annotatic
Linked Mc
Quick Diff
Spelling
*.c
*.h
*.htm
*.html
*.s
*.shtml
Add...
Remove

(4) Click the [Add...] button for [Associated editors:].
This displays the [Editor Selection] dialog box.


Editor Selection
Choose the editor for files of type (*.c)
● Internal Editors ○ External Programs
Assembly Editor
Default Compare Editor
External Search Editor
Internal Web Browser
Makefile Editor
No Source Found
Source Not Found Editor
Web Browser
Welcome
Browse...
OK
Cancel

(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.

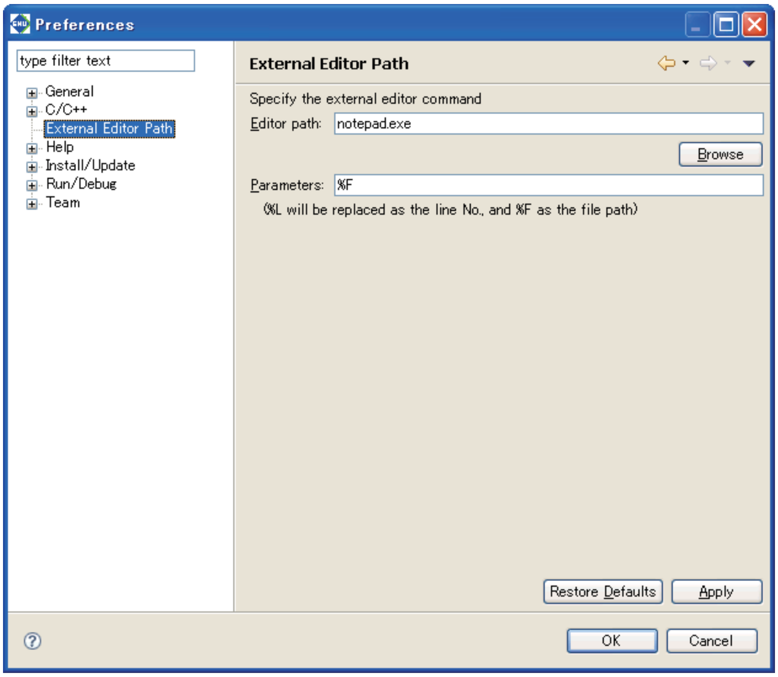
5.5.10 Launching External Editor by Specifying Line Number

External editor setting screen

If there are compiler errors in a project built with the IDE, the [Problems] view of the IDE displays a list of the files and line numbers that contain errors.
An external editor can be launched by selecting a line number in this error list.
The following describes the procedure for registering which external editor to launch.

(1) Select [Preferences...] from the [Window] menu.
The [Preferences] dialog box opens.

(2) Select [External Editor Path] in the tree list of setting items displayed on the left.


Preferences
type filter text
General
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
External Editor Path
Browse
Restore Defaults
Apply
OK
Cancel

Specify the external editor command
Editor path: notepad.exe
Parameters: %F
(%L will be replaced as the line No., and %F as the file path)

(3) Click the [Browse...] button and select the editor to launch.
Both path name and file name are limited to 255 characters. By default, "notepad.exe" (text editor) is set.

(4) Specify the editor startup parameters in [Parameters].
One of the following parameters can be set.
%F: Replaced by the file name during startup (cannot be omitted).
%L: Replaced by the line number during startup (can be omitted).

(5) Click the [OK] button to close the [Preferences] dialog box.

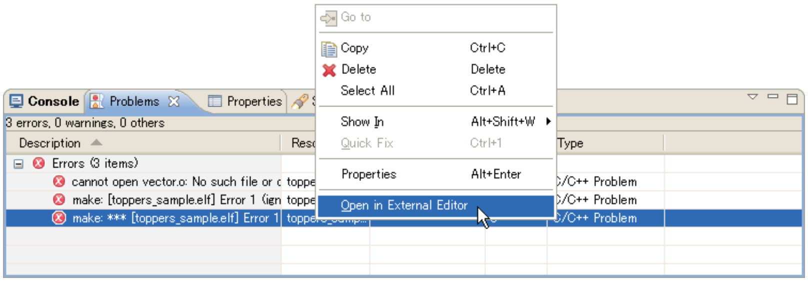
This establishes a correlation between the selected error and the external editor.

Starting up the external editor

If there are compiler errors in a project built with the IDE, the [Problems] view of the IDE displays a list of files and line numbers that contain errors.

The external editor can be launched by selecting a line number in this error list.

In the [Problems] view, right-click an error and select [Open in External Editor] in the context menu to launch the external editor set in [External Editor Path] in the [Preferences] dialog box.


Console
Problems
Properties
Description
Resc
Type
Errors (3 items)
Go to
Copy Ctrl+C
Delete Delete
Select All Ctrl+A
Show In Alt+Shift+W
Quick Fix Ctrl+1
Properties Alt+Enter
Open in External Editor

3 errors, 0 warnings, 0 others

cannot open vector.o: No such file or c toppe C/C++ Problem

make: [toppers_sample.elf] Error 1 (ign toppe C/C++ Problem

make: *** [toppers_sample.elf] Error 1 toppers_samp... C/C++ Problem

If the external editor does not launch due to incorrect settings or other reason, the error message "Error opening external editor" is displayed.

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/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.

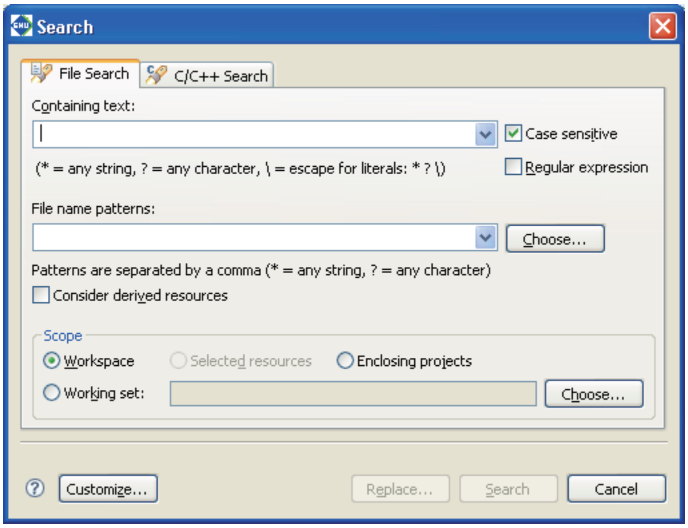
(2) Select [Text] from the [Search] menu and select a search domain (workspace, current project, current file or a specified working set) from its submenu.

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.


Search
File Search
C/C++ Search
Containing text:
☑ Case sensitive
(* = any string, ? = any character, \ = escape for literals: * ? \)
☐ Regular expression
File name patterns:
Choose...
Patterns are separated by a comma (* = any string, ? = any character)
☐ Consider derived resources
Scope
● Workspace
○ Selected resources
○ Enclosing projects
○ Working set:
Choose...
Customize...
Replace...
Search
Cancel

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 (*, \?, 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

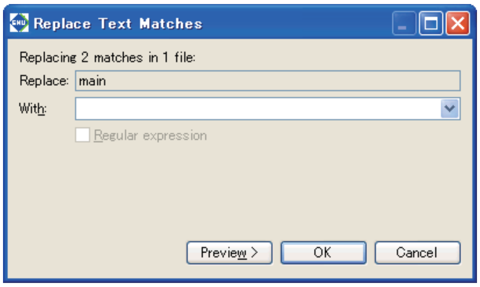
[Scope]

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.



Replacing 2 matches in 1 file:

Replace: main

With:

☐ Regular expression

[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.

[Preview]

Click to open the [Replace Text Matches] dialog box, which displays a list of matches. Check the replacement position.

[OK]

Replaces the currently selected occurrence of the search string with the new string entered and begins searching for the next occurrence.

[Cancel]

Stops the search.

[Search]

Performs a search using the parameters specified above.

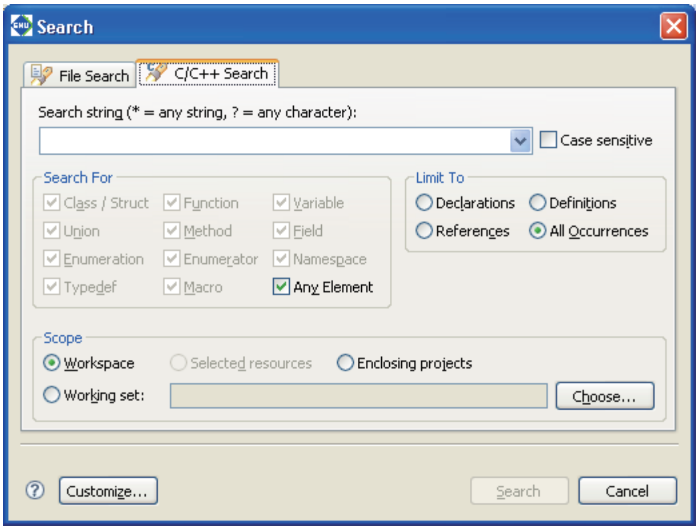
[Cancel]

Cancels a search.

5.6.3 C/C++ Search

Use C/C++ Search to search for strings, function names, and other elements within workspace resources. To perform a C/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.


Search
File Search
C/C++ Search
Search string (* = any string, ? = any character):
☐ Case sensitive
Search For
☑ Class / Struct ☑ Function ☑ Variable
☑ Union ☑ Method ☑ Field
☑ Enumeration ☑ Enumerator ☑ Namespace
☑ Typedef ☑ Macro ☑ Any Element
Limit To
○ Declarations ○ Definitions
○ References ● All Occurrences
Scope
● Workspace ○ Selected resources ○ Enclosing projects
○ Working set:
Choose...
Customize...
Search
Cancel

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]	Class or structure
[Function]	Global function or a function in name space (not including class, structure, and union member functions)
[Variable]	Variables (not including class, structure, and union members)
[Union]	Union
[Method]	Method (class, structure, or union members)
[Field]	Field (class, structure, or union members)
[Enumeration]	Enumeration
[Enumerator]	Enumerator
[Namespace]	Name space
[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.

[Declarations]	Declared location
[Definitions]	Defined place (function, method, variable, field)
[References]	Referenced location
[All Occurrences]	All occurrences, including the above

[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 that contains the resource selected in the [C/C++ Projects] or [Navigator] view
[Working Set:]	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/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.

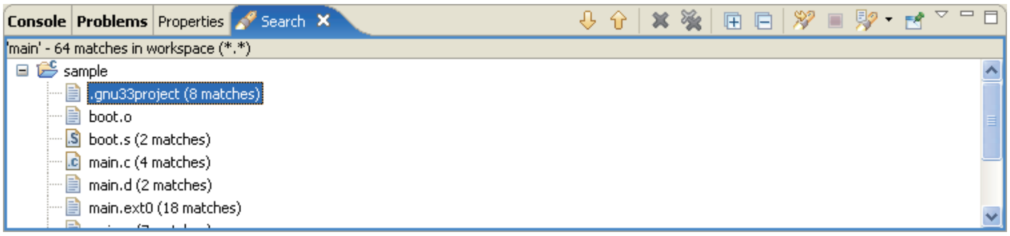
The search results are displayed in the [Search] view.

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, C/C++, and Text Searches are displayed in list form in [Search] view.



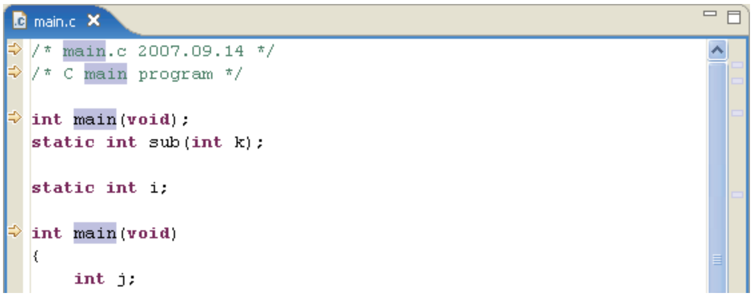
```
'main' - 64 matches in workspace (*.*)
⊟ sample
   .gnu33project (8 matches)
   boot.o
   boot.s (2 matches)
   main.c (4 matches)
   main.d (2 matches)
   main.ext0 (18 matches)
```

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.



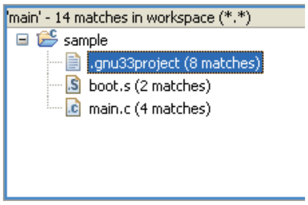
```
/* main.c 2007.09.14 */
/* C main program */

int main(void);
static int sub(int k);

static int i;

int main(void)
{
     int j;
```

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

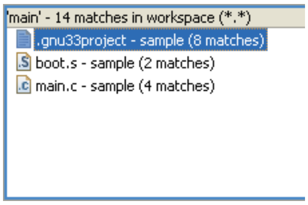
The [Search] view is initially set to display the search results in tree form. To display the search results in non-hierarchical mode, select [Flat Layout] from the [Show as List] view menu (▽).

Show as Tree



```
'main' - 14 matches in workspace (*.*)
⊟ sample
   .gnu33project (8 matches)
   boot.s (2 matches)
   main.c (4 matches)
```

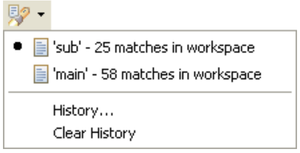
Show as List



```
'main' - 14 matches in workspace (*.*)
.gnu33project - sample (8 matches)
boot.s - sample (2 matches)
main.c - sample (4 matches)
```

Search history

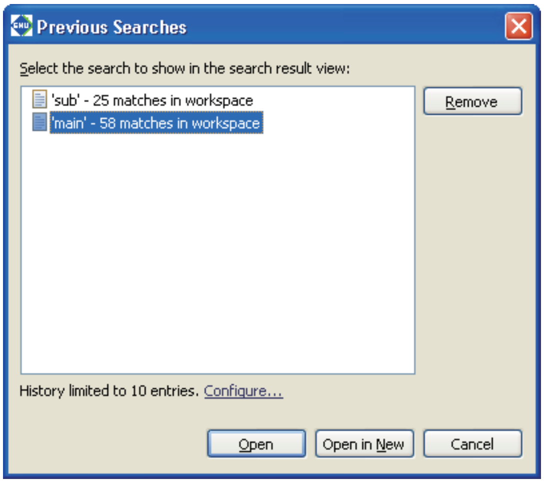
The [Show Previous Searches] shortcut in the [Search] view toolbar displays a list of the File, C/C++, and Text Searches previously performed.


'sub' - 25 matches in workspace
'main' - 58 matches in workspace
History...
Clear History

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.


Previous Searches
'sub' - 25 matches in workspace
'main' - 58 matches in workspace
Remove
Open
Open in New
Cancel

Select the search to show in the search result view:

History limited to 10 entries. Configure...

[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.

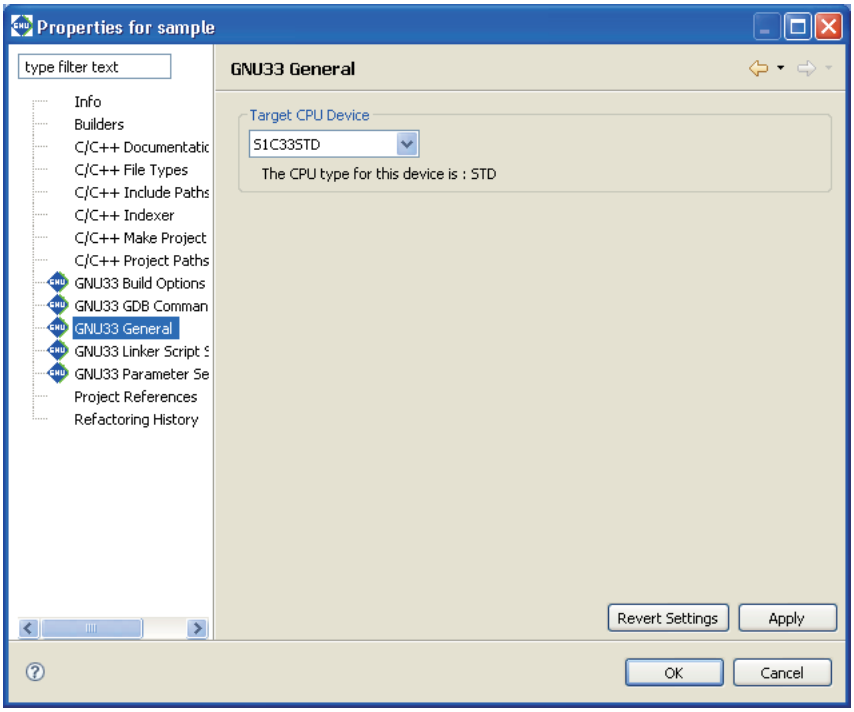
5.7.1 Setting the CPU Type

The startup command options for tools and emulation libraries to be linked depend on the type of the processor for which you are developing the application. You must select the correct CPU type before attempting a build process.
In most cases, you will not need to select a CPU type, since this would presumably have been done when you created the project. If necessary, you can reset the CPU type as follows:

(1) In the [C/C++ Projects] or [Navigator] view, select a project for which you want to change the CPU type.

(2) Select [Properties] from the [Project] menu or from the context menu in the above view.
This displays the [Properties] dialog box.

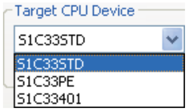
(3) Select [GNU33 General] from the properties list.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 General
Revert Settings
Apply
OK
Cancel

Target CPU Device
S1C33STD
The CPU type for this device is : STD

The [Target CPU Device] combo box shows the currently selected target CPU type (the default type is S1C33STD).

(4) Select the target CPU type from the dropdown list in the [Target CPU Device] combo box.


Target CPU Device
S1C33STD
S1C33STD
S1C33PE
S1C33401

S1C33STD: When the target processor is a device with the C33 STD Core embedded
S1C33PE: When the target processor is a device with the C33 PE Core embedded
S1C33401: When the target processor is the S1C33401

(5) 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 and closes the dialog box.
[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 5.7.9) 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).

The table below lists the settings changed based on the selection made here.

Table 5.7.1.1 Option settings by CPU type

CPU type	Affected settings	Effects
S1C33STD	Build option settings	`-mc33pe` and `-mc33adv` are removed from the options. The **IDE** switches to C33 STD Core emulation libraries.
	Linker script settings	The **IDE** switches to C33 STD Core emulation libraries.
	Parameter map settings	The CPU type is not output when the parameter file is output.
	gdb startup command settings	Settings in the template are reset to the default C33 STD Core values. (ICD6 + USB connection) The edited commands in the [GNU33 GDB Commands] page are not updated.
S1C33PE	Build option settings	`-mc33pe` is added to the options. The **IDE** switches to C33 PE Core emulation libraries.
	Linker script settings	The **IDE** switches to C33 PE Core emulation libraries.
	Parameter map settings	The CPU type (`C33_PE`) is output when the parameter file is output.
	gdb startup command settings	Settings in the template are reset to the default C33 PE Core values. (ICD6 + USB connection) The edited commands in the [GNU33 GDB Commands] page are not updated.
S1C33401	Build option settings	`-mc33adv` and `-mc33401` are added to the options. The **IDE** switches to S1C33401 emulation libraries.
	Linker script settings	The **IDE** switches to S1C33401 emulation libraries.
	Parameter map settings	The CPU type (`C33_ADVANCED`) is output when the parameter file is output.
	gdb startup command settings	Settings in the template are reset to the default S1C33401 values. (ICD6 + USB connection) The edited commands in the [GNU33 GDB Commands] page are not updated.

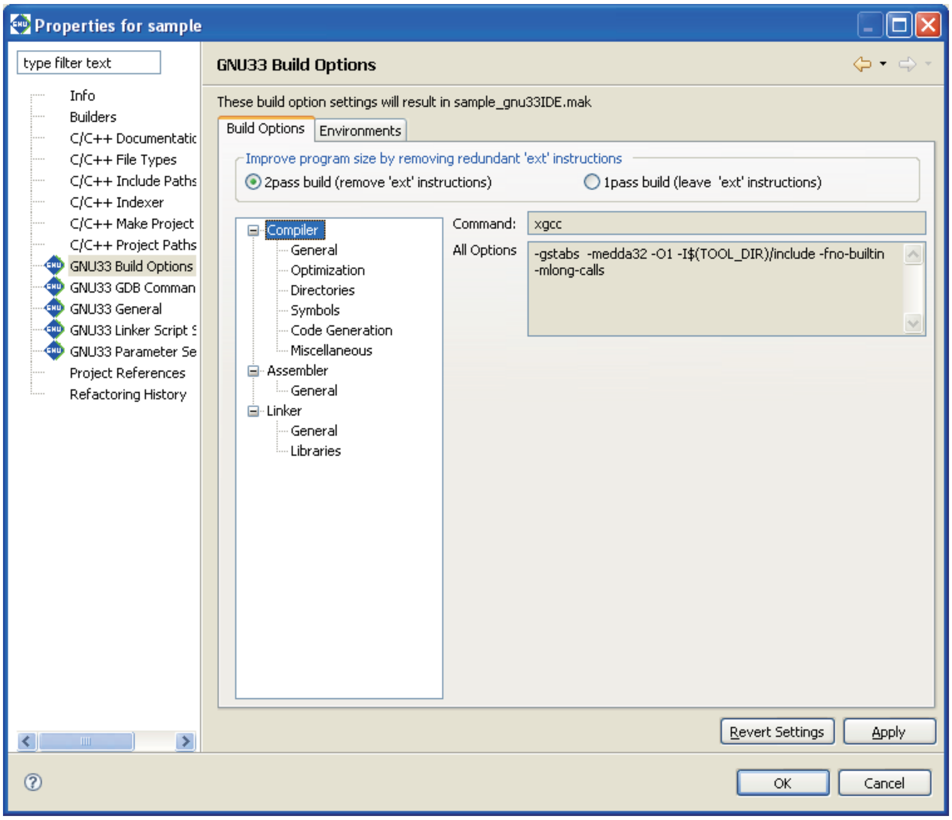
5.7.2 Selecting Two-Pass/One-Pass Make

The two-pass make process executes the assembler again to remove redundant `ext` instructions after the first linkage has resolved the symbol address information. For example, the `call` instructions with the `ext` instructions that are expanded from function calls by compiling and assembling may include unnecessary `ext` instruction(s) depending on the branch distance. Removing them can reduce the program size. For details, see Section 8.8, "Optimization of Extended Instructions". The **IDE** is capable of creating two make files, normal one-pass makefile and two-pass makefile. Either one can be selected by the following procedure:

(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 [GNU33 Build Options] from the properties list.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

Command: xgcc

All Options: -gstabs -medda32 -O1 -I$(TOOL_DIR)/include -fno-builtin -mlong-calls

(4) Select a build process using the radio buttons in the [Improve program size by removing redundant 'ext' instructions] field.

[2pass build] (default: ON)

Selecting this button will generate a makefile that contains the two-pass make process to optimize the extended instructions. This increases the build time but reduces the program size.

[1pass build]

Selecting this button will generate a makefile that contains a one-pass make process that does not remove the ext instructions.

(5) Click the [Apply] button to change other properties or the [OK] button to complete property settings.
If settings in a [GNU33 Build Options] page have been changed, a dialog box appears for selecting "clean" build (see Section 5.7.9) 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 to the state in which the page was opened.

5.7.3 Setting Compiler Options

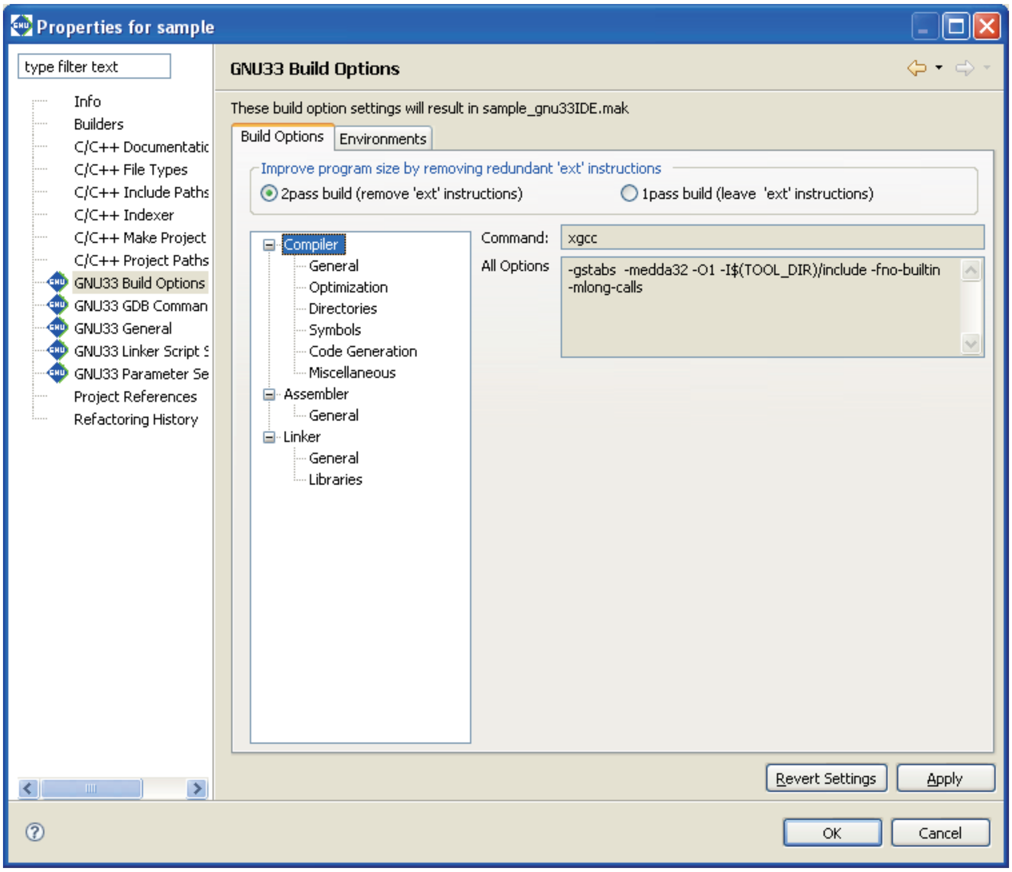
Do the following to set C/C++ compiler command options.

(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 [GNU33 Build Options] from the properties list.

(4) Select [Compiler] from the [Build Options] tree.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

Command: xgcc

All Options: -gstabs -medda32 -O1 -I$(TOOL_DIR)/include -fno-builtin -mlong-calls

The [Command:] field shows the name of the C/C++ compiler. The [All Options] column lists currently set compiler options.

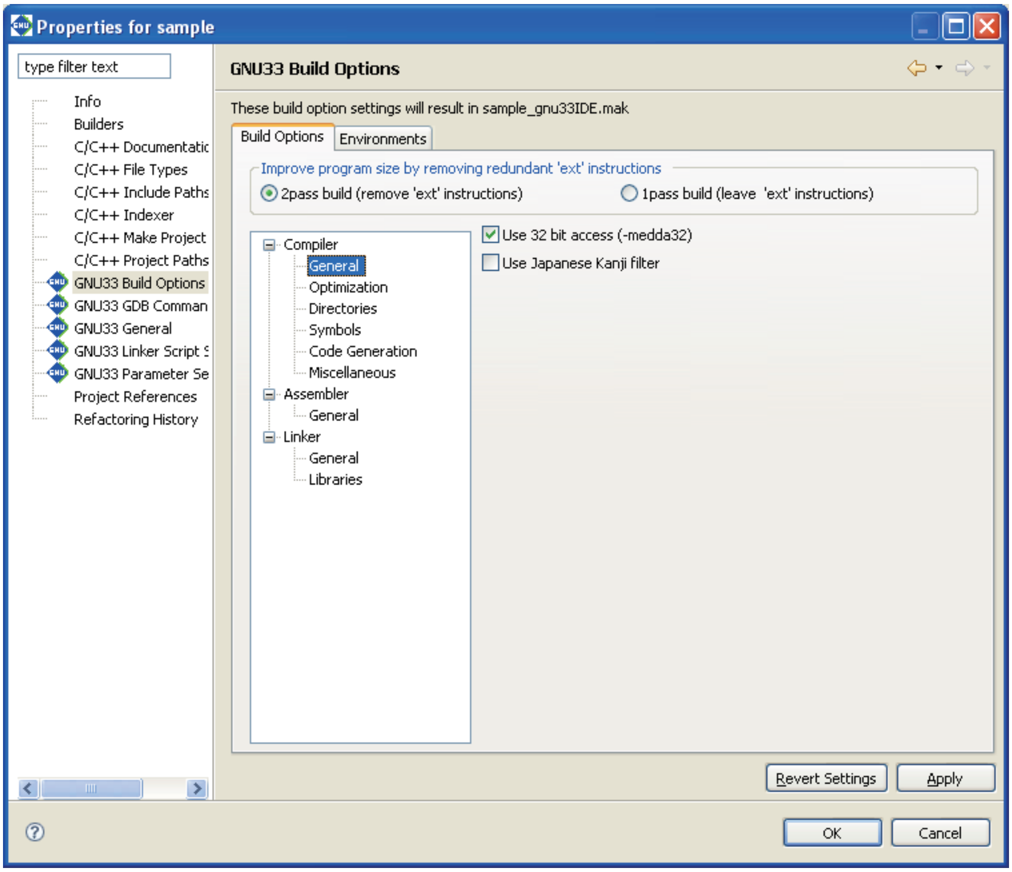
(5) Select a category from the [Compiler] tree list and set the necessary options.

(6) Click the [Apply] button to change other properties or the [OK] button to complete property settings.
If settings in a [GNU33 Build Options] page have been changed, a dialog box appears for selecting "`clean`" build (see Section 5.7.9) 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 to the state in which the 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/C++ compiler.

[General]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

☑ Use 32 bit access (-medda32)

☐ Use Japanese Kanji filter

Select basic compiler options from this page.

[Use 32 bit access (-medda32)] (default: ON)

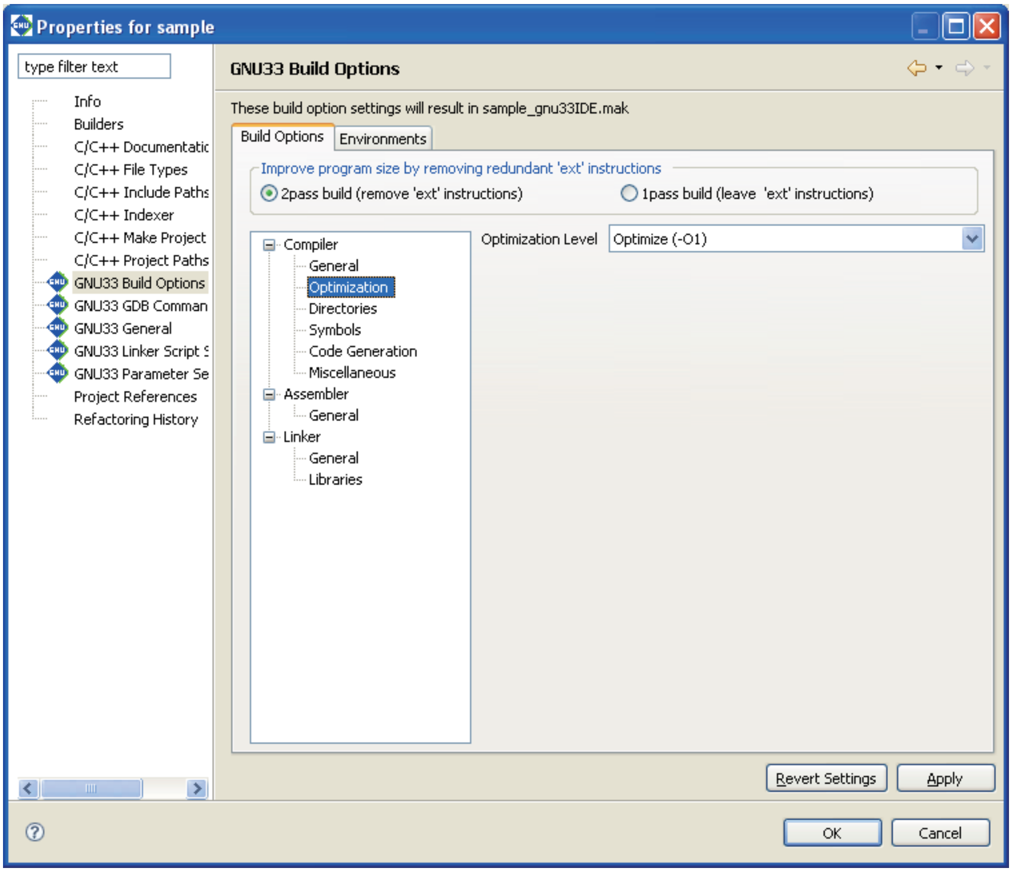
If this option is specified, the compiler outputs code for 32-bit access without using the default data area. This option cannot be specified in combination with the `-fPIC` option (in [Code Generation] page).

[Use Japanese Kanji filter]

If this option is enabled, Shift JIS codes in the source will be read appropriately during compilation. Disabling the option is equivalent to specifying the `-mno-sjis-filt` option when calling a compiler, in which case the above processing is not performed.

The default status of the checkbox depends on the language of the OS used to run the IDE. The checkbox is selected by default if the IDE is launched in a Japanese-language OS environment; in other language versions, the checkbox is unselected by default. (When the checkbox is unselected, the `-mno-sjis-filt` option is specified during compilation.)

[Optimization]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

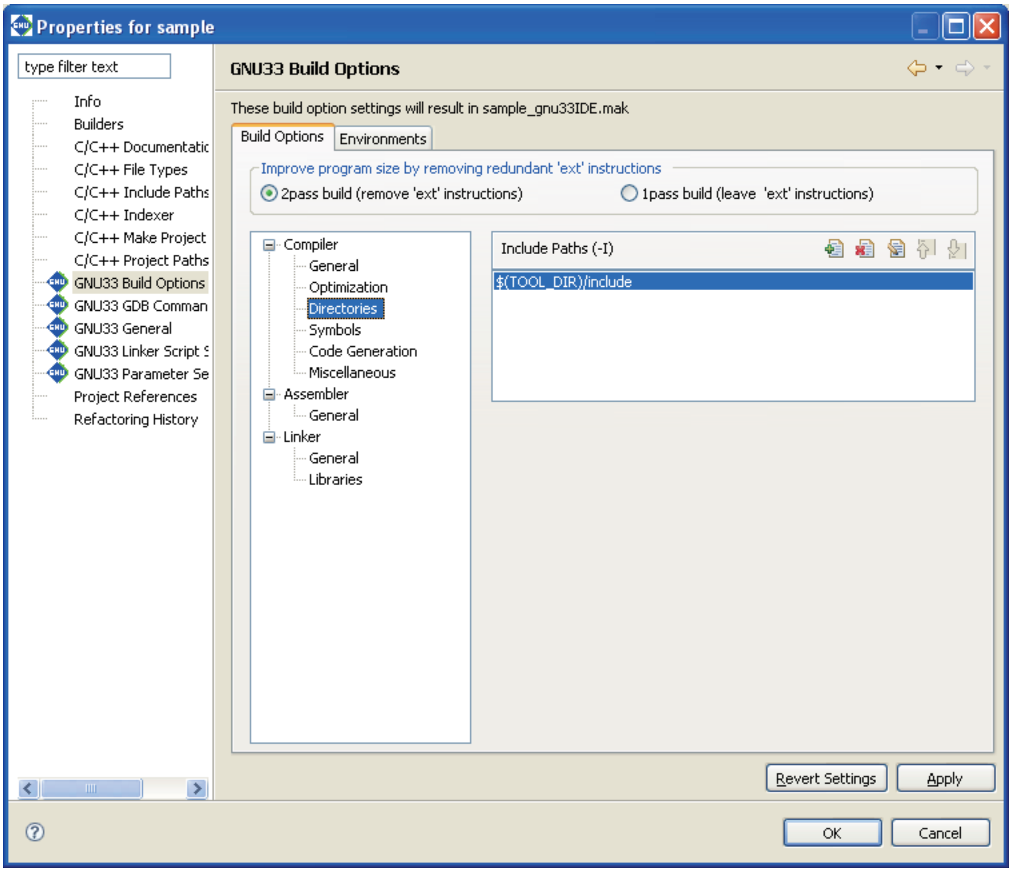
Optimization Level: Optimize (-O1)

Select compiler optimization options from this page.

[Optimization Level] (default: `-O1`)

Select an optimization level (`-O0` to `-O3`, `-Os`).

[Directories]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Include Paths (-I)
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

$(TOOL_DIR)/include

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 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 gnu33 tool directory is defined. The defined contents can be verified in the [Environments] tab page.

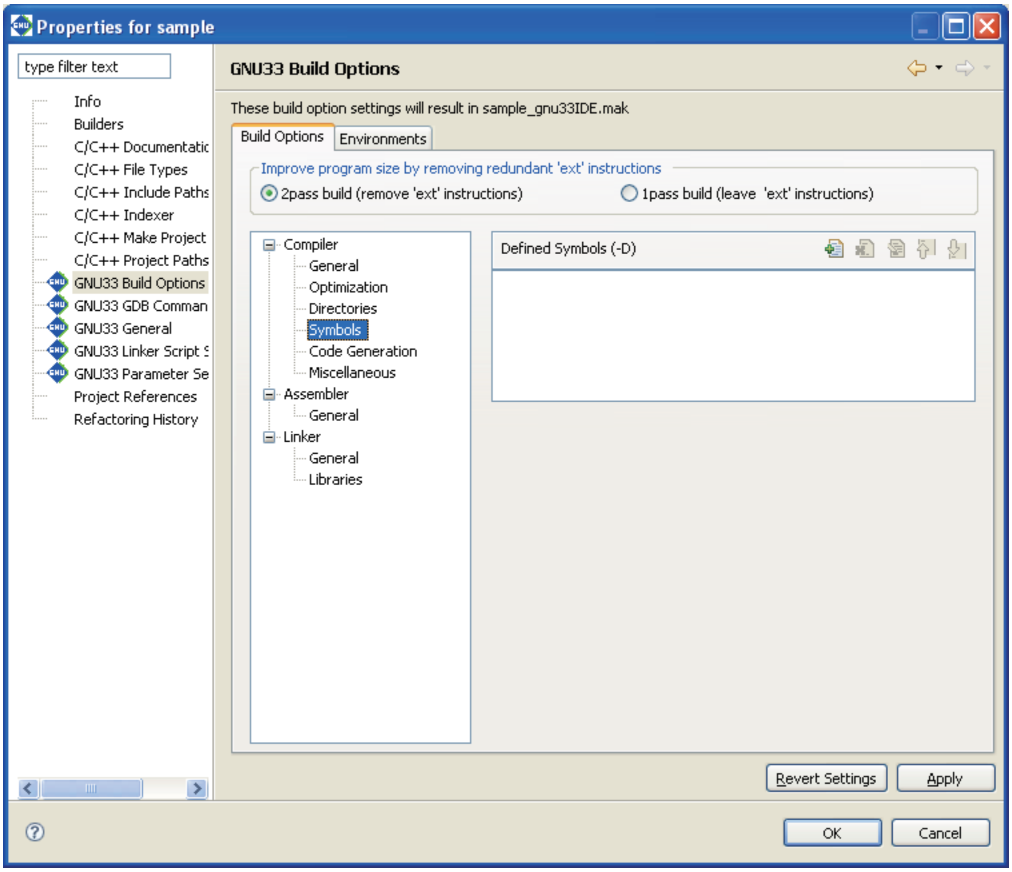
Example: If the gnu33 tools have been installed in the c:\EPSON\gnu33 directory

```
TOOL_DIR = c:/EPSON/gnu33
```

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/gnu33/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]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Defined Symbols (-D)
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

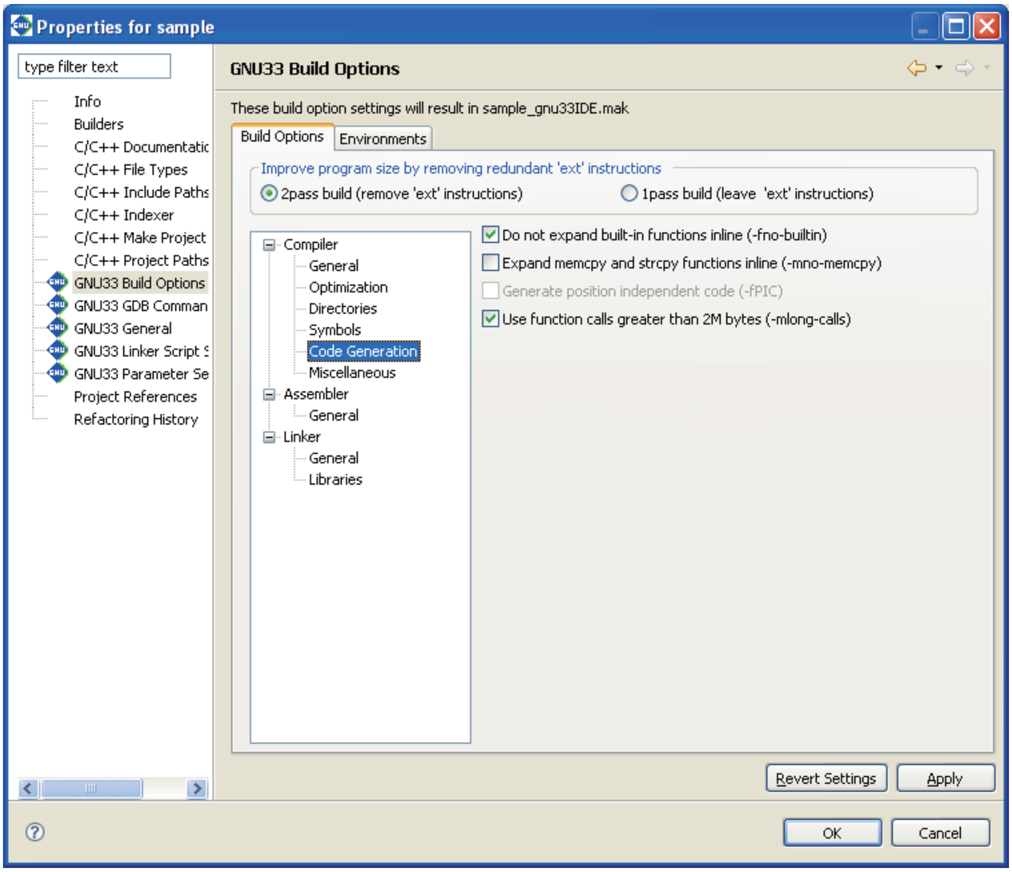
Set compiler macro-definition options from this page.

[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. *<macro-name>* or *<macro-name>*=*<replacement string>*
[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]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

☑ Do not expand built-in functions inline (-fno-builtin)

☐ Expand memcpy and strcpy functions inline (-mno-memcpy)

☐ Generate position independent code (-fPIC)

☑ Use function calls greater than 2M bytes (-mlong-calls)

Select compiler code generation options 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".

[Expand memcpy and strcpy functions inline (-mno-memcpy)] (default: OFF)

If this option is specified, the `memcpy()` and `strcpy()` functions are expanded in-line.

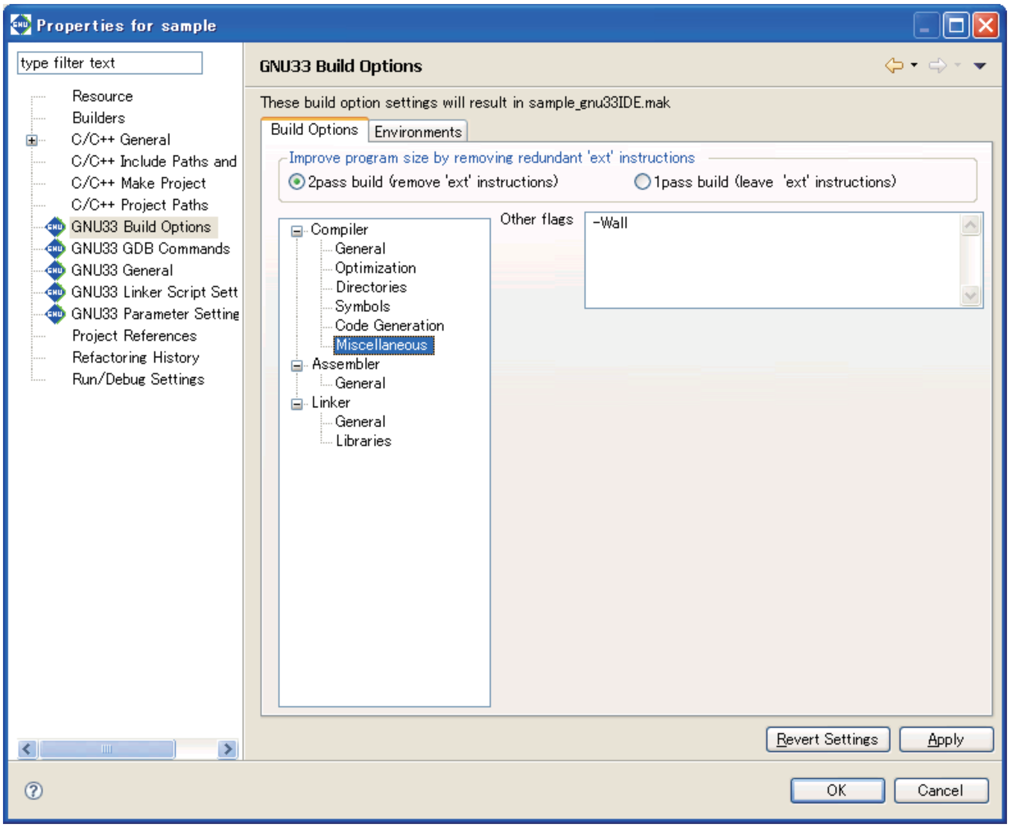
[Generate position independent code (-fPIC)] (default: OFF)

Specifying this option generates position-independent code.
This option cannot be specified in combination with the `-medda32` option (in the [General] page).

[Use function calls greater than 2M bytes (-mlong-calls)] (default: ON)

Function calls 2M bytes or more apart from the current position are executed.

[Miscellaneous]


Properties for sample
type filter text
Resource
Builders
C/C++ General
C/C++ Include Paths and
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Commands
GNU33 General
GNU33 Linker Script Sett
GNU33 Parameter Setting
Project References
Refactoring History
Run/Debug Settings
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

Other flags: -Wall

Set other compiler options from this page.

[Other flags] (default: -Wall)

Enter other options directly into this text field. Separate each option with one or more spaces.
The `-mc33adv` and `-mc33401` options are also specified here if the CPU type selected is the "S1C33401". The `-mc33pe` option is specified here if the CPU type selected is the "S1C33PE".
When the two-pass make process is selected, the `-S` option is automatically specified for compilation and an assembly file (extension: .ext0) is output, and this file is assembled by the assembler.
It is not necessary to specify the compiler `-S` option or compiler `-c` option in [Other flags].
The assembled results of the C source can be confirmed in the < C SOURCE FILE NAME.ext0 > file.
Also the `-gstabs` option is automatically specified.

5.7.4 Setting Assembler Options

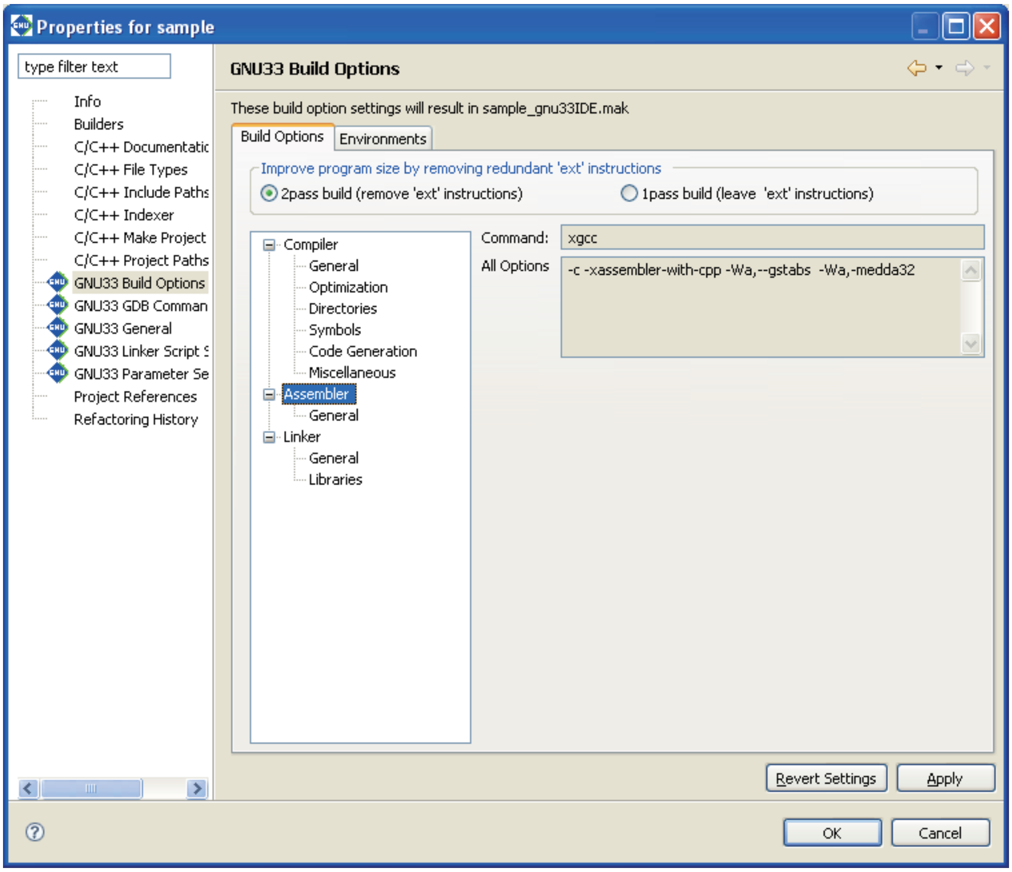
Do the following to set assembler command options.

(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 [GNU33 Build Options] from the properties list.

(4) Select [Assembler] from the [Build Options] tree.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions
● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

Command: xgcc

All Options: -c -xassembler-with-cpp -Wa,--gstabs -Wa,-medda32

The [Command:] field shows the program name of the compiler*, and the [All Options] column lists the currently set options.

Note: When the `-c -xassembler-with-cpp` option is specified, the **IDE** assembles the assembler source using the specified C compiler. It is not necessary to specify the compiler `-c` option or `-xassembler-with-cpp` option to All Options.

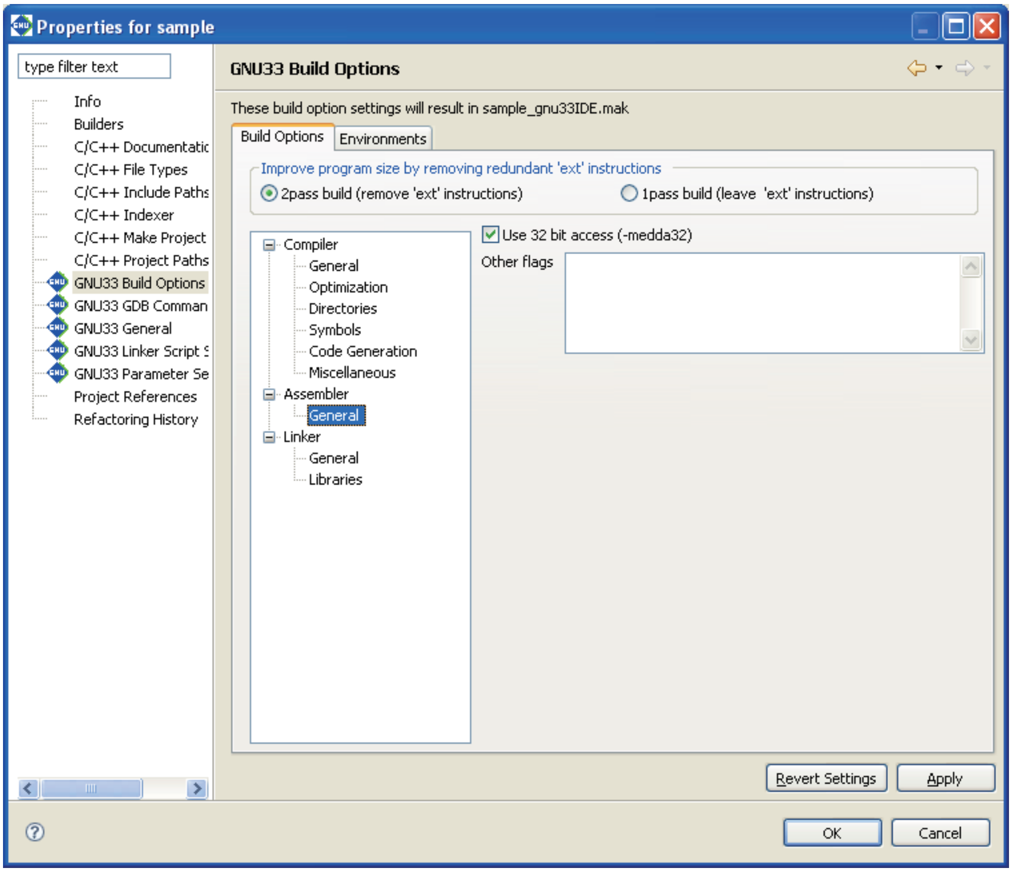
(5) Select [General] from the [Assembler] tree list and set the necessary options.

(6) Click the [Apply] button to change other properties or the [OK] button to complete property settings.
If settings in a [GNU33 Build Options] page have been changed, a dialog box appears for selecting "`clean`" build (see Section 5.7.9) 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 to the state in which the 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]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions)   ○ 1pass build (leave 'ext' instructions)

☑ Use 32 bit access (-medda32)

Other flags

Select assembler options from this page. The `-c`, `-xassembler-with-cpp`, and `-Wa,--gstabs` options are always added, no matter how the following are set:

[Use 32 bit access (-medda32)] (default: ON)

If this option is specified, the assembler outputs code for 32-bit access without using the default data area.

[Other flags] (default: none)

Enter the options to be passed to the assembler. Insert one or more spaces between each option.
The `-mc33adv` and `-mc33401` options are also specified here if the CPU type selected is the "S1C33401". Similarly, the `-mc33pe` option is specified here if the CPU type is selected is the "S1C33PE".
The options entered are passed to the assembler as "`-Wa,`*<option>*`,` ...".

5.7.5 Setting Linker Options

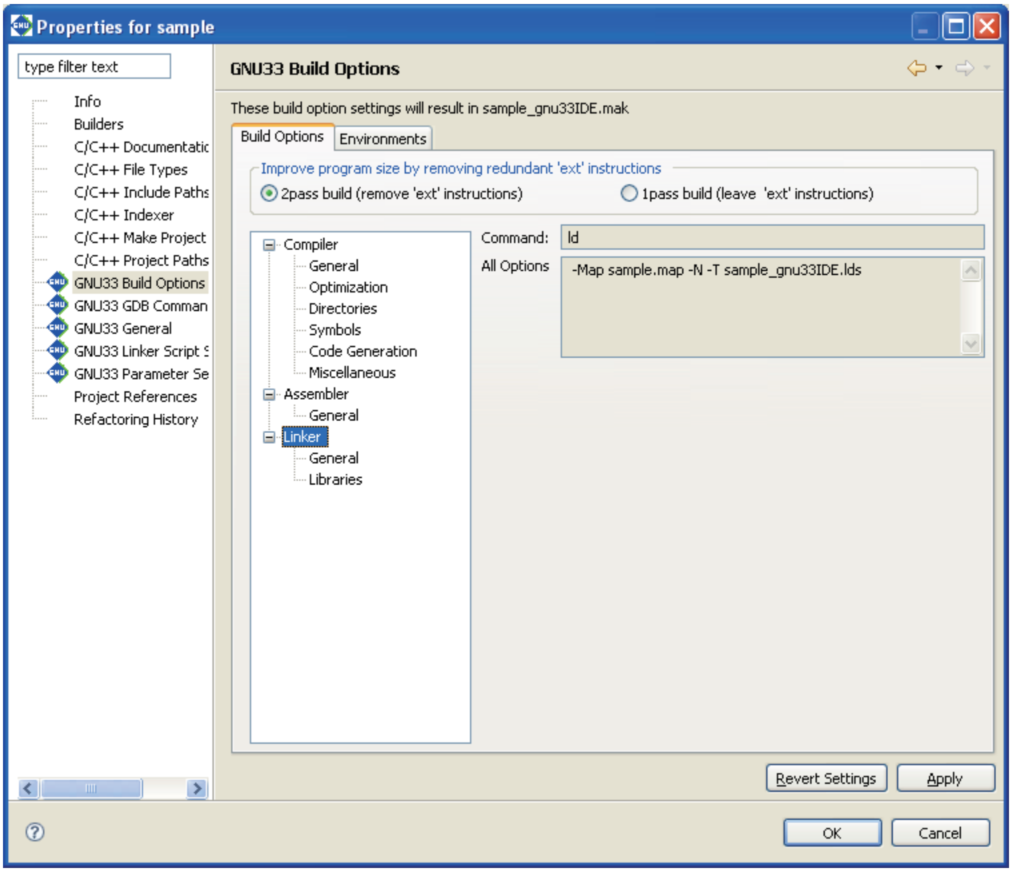
Do the following to set the linker command options:

(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 [GNU33 Build Options] from the properties list.

(4) Select [Linker] from the [Build Options] tree.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

Command: ld

All Options: -Map sample.map -N -T sample_gnu33IDE.lds

The [Command:] field shows the program name of the linker. The [All Options] column lists the currently set options.

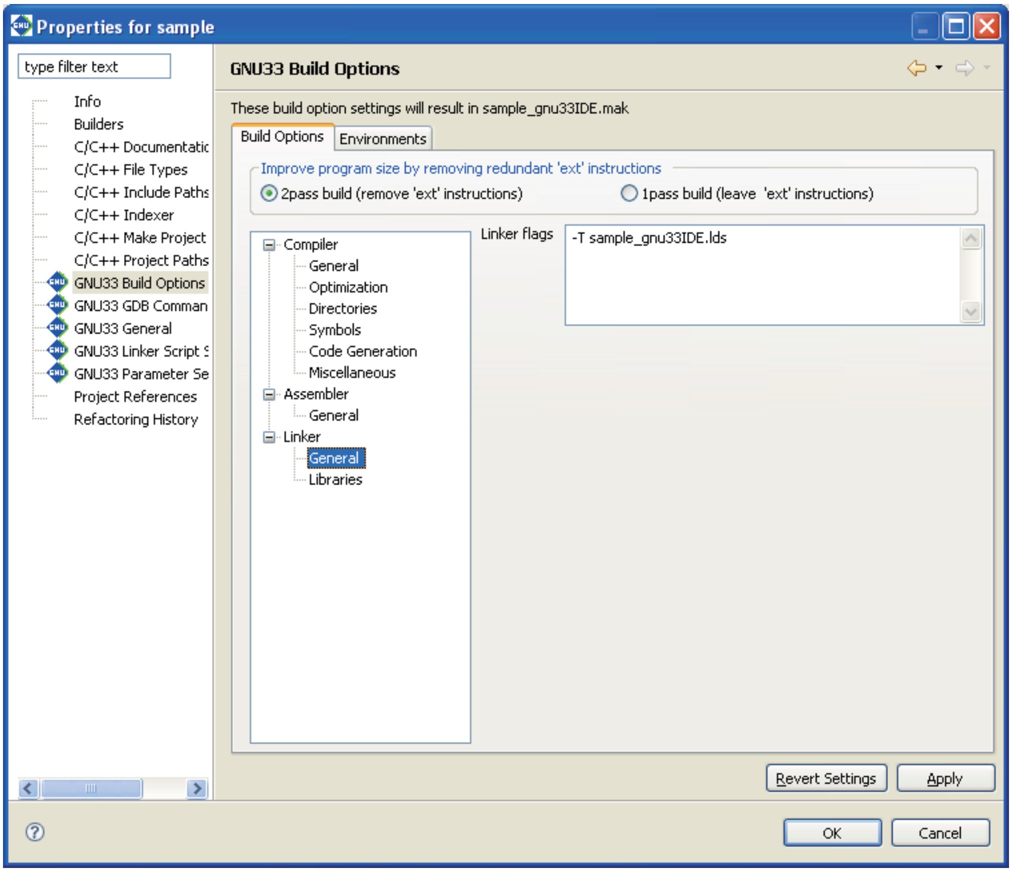
(5) Select a category from the [Linker] tree list and set the necessary options.

(6) Click the [Apply] button to change other properties or the [OK] button to complete property settings.
If settings in a [GNU33 Build Options] page have been changed, a dialog box appears for selecting "`clean`" build (see Section 5.7.9) 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 the 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]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Improve program size by removing redundant 'ext' instructions
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

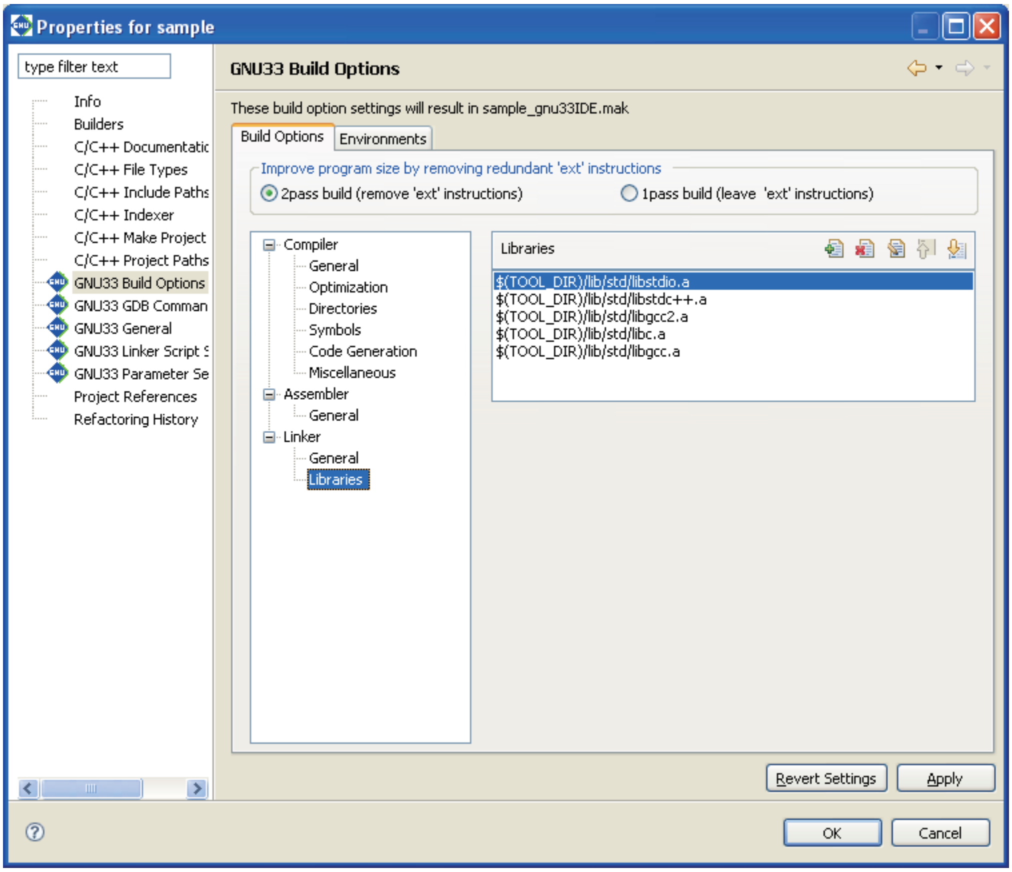
Linker flags: -T sample_gnu33IDE.lds

The `-Map` and `-N` options are always added. Set other linker options from this page.

[Linker flags] (default: `-T<`*project name*`>_gnu33IDE.lds`)

Enter other linker options in this text field. Insert one or more spaces between each option.

[Libraries]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

```
$(TOOL_DIR)/lib/std/libstdio.a
$(TOOL_DIR)/lib/std/libstdc++.a
$(TOOL_DIR)/lib/std/libgcc2.a
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a
```

Set the libraries to be linked from this page.

[Libraries] (default: `libstdio.a`, `libstdc++.a`, `libgcc2.a`, `libc.a`, `libgcc.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.

∗ About `$(TOOL_DIR)`

The [Libraries] column lists "`$(TOOL_DIR)/lib/std/libxxx.a`" that is set by default.

`$(`*environment variable*`)` is 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 gnu33 tool directory is defined. The defined contents can be verified in the [Environments] tab page.

Example: If the gnu33 tools have been installed in the c:\EPSON\gnu33 directory

```
TOOL_DIR = c:/EPSON/gnu33
```

Since the macro is replaced with the contents of the environment variable described in `()` during execution of **make.exe**, `-I$(TOOL_DIR)/lib/std/libxxx.a` will be resolved to `-Ic:/EPSON/gnu33/lib/std/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.

5.7.6 Generated Makefile

Building a project generates a makefile named "<*project name*>`_gnu33IDE.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 Gnu33 Plug-in for Eclipse
# This file should be placed directly under the project folder

# macro definitions for target file                                                  (1)
TARGET= sample
GOAL= $(TARGET).elf

# macro definitions for tools                                                        (2)
TOOL_DIR= C:/EPSON/GNU33
CC= $(TOOL_DIR)/xgcc
CXX= $(TOOL_DIR)/xgcc
AS= $(TOOL_DIR)/xgcc
AS_CC= $(TOOL_DIR)/as
LD= $(TOOL_DIR)/ld
RM= $(TOOL_DIR)/rm
SED= $(TOOL_DIR)/sed
CP= $(TOOL_DIR)/cp
OBJDUMP= $(TOOL_DIR)/objdump

# macro definitions for tool flags                                                   (3)
CFLAGS= -B$(TOOL_DIR)/ -gstabs -S  -medda32  -O1 -I$(TOOL_DIR)/include -fno-builtin
-mlong-calls  -Wall
CXXFLAGS= ${CFLAGS}
ASFLAGS= -B$(TOOL_DIR)/ -c -xassembler-with-cpp -Wa,--gstabs  -Wa,-medda32
ASFLAGS_CC=  -medda32
LDFLAGS=  -Map sample.map -N -T sample_gnu33IDE.lds
ALLOBJDUMP_FILE= __allobjects
EXTFLAGS= -Wa,-mc33_ext -Wa,$(TARGET).dump -Wa,$(ALLOBJDUMP_FILE).dump
EXTFLAGS_CC= -mc33_ext $(TARGET).dump $(ALLOBJDUMP_FILE).dump
OBJDUMPFLAGS= -t

# macro for switching 2pass or 1pass build                                           (4)
PASS= 2pass

# search paths for source files
vpath %.c
vpath %.cpp
vpath %.cc
vpath %.cxx
vpath %.C
vpath %.s

# macro definitions for object files                                                 (5)
OBJS= boot.o \
      main.o \

# macro definitions for library files                                                (6)
OBJLDS= $(TOOL_DIR)/lib/std/libstdio.a \
        $(TOOL_DIR)/lib/std/libstdc++.a \
        $(TOOL_DIR)/lib/std/libgcc2.a \
        $(TOOL_DIR)/lib/std/libc.a \
        $(TOOL_DIR)/lib/std/libgcc.a \

# macro definitions for assembly files generated from c source files
CEXTTEMPS= main.ext0 \                                                               (7)

# macro definitions for dependency files                                             (8)
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                                    (9)
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                                                  (10)
.PHONY : all clean

all : $(GOAL)
                                                                            (11)
$(TARGET).elf : $(OBJS) sample_gnu33IDE.mak sample_gnu33IDE.lds
ifeq ($(PASS), 1pass)
# 1pass linking
  $(LD) $(LDFLAGS) -o $@ $(OBJS) $(OBJLDS)
else
# 1pass linking
  -$(LD) $(LDFLAGS) -o $@ $(OBJS) $(OBJLDS) 2>lderr
  @if [ -s lderr ]; then \
         cmd /c "type lderr" \
         && $(RM) -f $(TARGET).elf \
         && exit 1; \
  else $(RM) -f lderr ; \
  fi
  $(OBJDUMP) $(OBJDUMPFLAGS) $@ > $(TARGET).dump
  $(RM) -f $(TARGET).elf
# create all objets dump file
  $(RM) -f $(ALLOBJDUMP_FILE).dump
  for NAME in $(OBJS) ; do \
         $(OBJDUMP) $(OBJDUMPFLAGS) $$NAME >> $(ALLOBJDUMP_FILE).dump  ; done
# save 1pass object files
  @if [ -e obj1pass ]; then \
         cmd /c "rd /s /q obj1pass" ; \
  fi
  cmd /c "md obj1pass"
  for NAME in $(subst /,\\,$(OBJS)) ; do \
         cmd /c "copy /y $$NAME obj1pass\\$$NAME" >nul ; done \
  && $(RM) -f $(OBJS)
# 2pass for assembly files
  $(AS) $(ASFLAGS) $(EXTFLAGS) -o boot.o boot.s
# 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)
# restore 1pass object files
  $(RM) -f $(OBJS) \
  && \
  for NAME in $(subst /,\\,$(OBJS)) ; do \
         cmd /c "copy /y obj1pass\\$$NAME $$NAME" >nul ; done \
  && cmd /c "rd /s /q obj1pass"
endif
  @cmd /c "echo ---------------- Finished building target : $@ ----------------"

## boot.s
                                                                            (12)
boot.o : boot.s
  $(AS) $(ASFLAGS) -o $@ $<
  $(DEPCMD_AS)

## main.c
main.o : main.c main.ext0
  $(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<
  $(AS_CC) $(ASFLAGS_CC) -o $@ $(@:%.o=%.ext0)
  $(DEPCMD_CC)

# dependecies for assembled c source files
main.ext0 : main.c
```

```
# include dependency files
-include $(DEPS)
# clean files
                                                                          (13)
clean :
  $(RM) -f $(OBJS) $(TARGET).elf $(TARGET).map $(DEPS) $(CEXTTEMPS) $(TARGET).dump
$(ALLOBJDUMP_FILE).dump lderr
  @if [ -e obj1pass ]; then \
          cmd /c "rd /s /q obj1pass" ; \
  fi
```

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 S5U1C33001C 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 ([GNU33 Build Options]).

(4) Defines the 2-pass/1-pass build option. The option reflects the selected content of project properties ([GNU33 Build Options]).

(5) The object file names corresponding to the source files in the project are written here following "`OBJS=`".
The contents written in this field change when source files are added or deleted.

(6) The library file names set in [GNU33 Build Options] > [Build Options] > [Linker] > [Libraries] are written here following "`OBJLDS=`" in the same way as for "`OBJS=`".

(7) 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/C++ source files have a file extension "`.ext0`".

(8) 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.

(9) Defines the execution commands to be stored in a dependency list.

(10) Defines the target.

(11) This is the dependency list for the target to be built and executable format object files.
When the two-pass make process is selected, the executable format object file will be generated after optimizing the extended instructions.

(12) This is the dependency list for object files generated from each source.
Adding or deleting source files will change the information in this field.

(13) 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".

5.7.7 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*>`_gnu33IDE.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 3.7, "Data Area and Sections", and Chapter 9, "Linker".

Linker script setup page

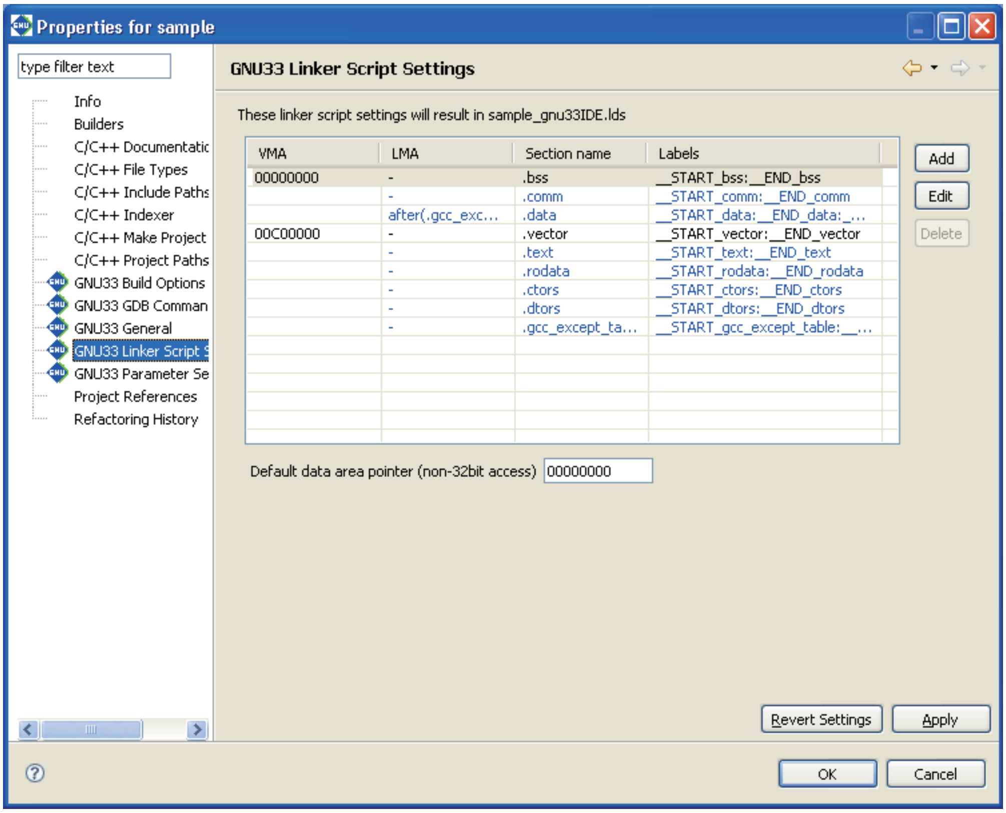
Use the [GNU33 Linker Script Settings] page of project properties to set a linker script.

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 [GNU33 Linker Script Settings] from the properties list.

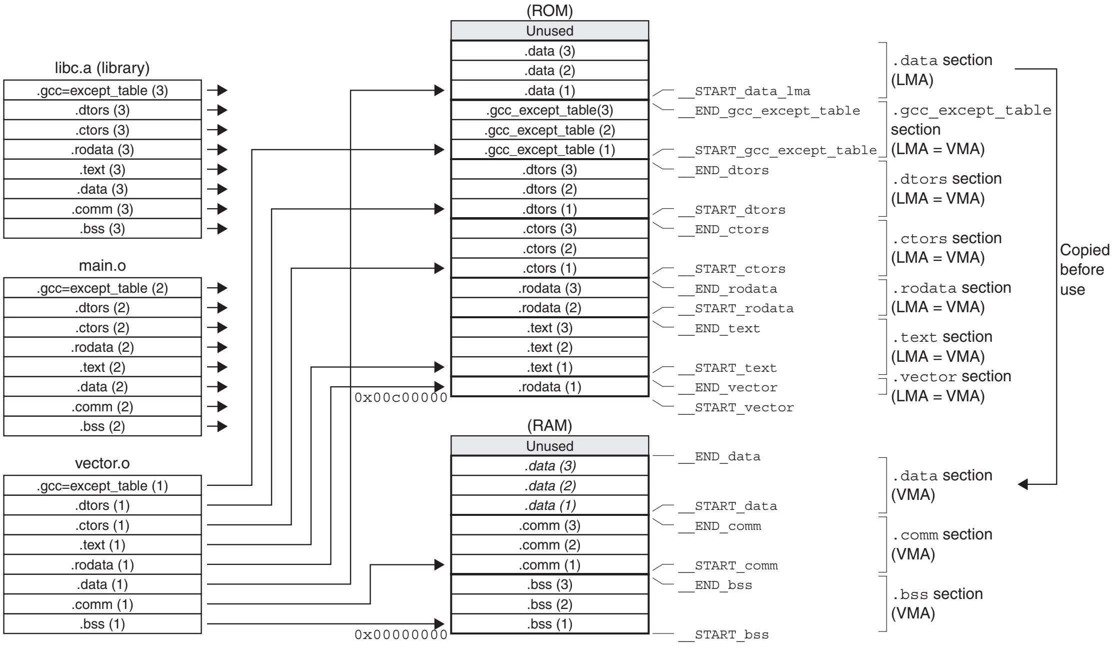

Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Linker Script Settings
Add
Edit
Delete
Revert Settings
Apply
OK
Cancel

These linker script settings will result in sample_gnu33IDE.lds

VMA	LMA	Section name	Labels
00000000	-	.bss	__START_bss:__END_bss
	-	.comm	__START_comm:__END_comm
	after(.gcc_exc...	.data	__START_data:__END_data:_...
00C00000	-	.vector	__START_vector:__END_vector
	-	.text	__START_text:__END_text
	-	.rodata	__START_rodata:__END_rodata
	-	.ctors	__START_ctors:__END_ctors
	-	.dtors	__START_dtors:__END_dtors
	-	.gcc_except_ta...	__START_gcc_except_table:__...

Default data area pointer (non-32bit access) 00000000

The list shows the configuration and location of sections in an execution file (.elf). This memory map is shown in Figure 5.7.7.1.


(ROM)
Unused
.data (3)
.data (2)
.data (1)
.gcc_except_table(3)
.gcc_except_table (2)
.gcc_except_table (1)
.dtors (3)
.dtors (2)
.dtors (1)
.ctors (3)
.ctors (2)
.ctors (1)
.rodata (3)
.rodata (2)
.text (3)
.text (2)
.text (1)
.rodata (1)
0x00c00000
libc.a (library)
.gcc=except_table (3)
.dtors (3)
.ctors (3)
.rodata (3)
.text (3)
.data (3)
.comm (3)
.bss (3)
main.o
.gcc=except_table (2)
.dtors (2)
.ctors (2)
.rodata (2)
.text (2)
.data (2)
.comm (2)
.bss (2)
vector.o
.gcc=except_table (1)
.dtors (1)
.ctors (1)
.text (1)
.rodata (1)
.data (1)
.comm (1)
.bss (1)
__START_data_lma
__END_gcc_except_table
__START_gcc_except_table
__END_dtors
__START_dtors
__END_ctors
__START_ctors
__END_rodata
__START_rodata
__END_text
__START_text
__END_vector
__START_vector
.data section
(LMA)
.gcc_except_table
section
(LMA = VMA)
.dtors section
(LMA = VMA)
.ctors section
(LMA = VMA)
.rodata section
(LMA = VMA)
.text section
(LMA = VMA)
.vector section
(LMA = VMA)
Copied
before
use
(RAM)
Unused
.data (3)
.data (2)
.data (1)
.comm (3)
.comm (2)
.comm (1)
.bss (3)
.bss (2)
.bss (1)
0x00000000
__END_data
__START_data
__END_comm
__START_comm
__END_bss
__START_bss
.data section
(VMA)
.comm section
(VMA)
.bss section
(VMA)

(when vector.o, main.o, and libc.a are linked)

Figure 5.7.7.1 Selection location in default settings

As shown in the [Section name] column, the following nine basic sections are set in advance:

`.bss`: A section in which variables without initial values are placed. (This is normally located in RAM.)

`.comm`: 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.)

`.ctors`: Pointer arrays to global class constructor functions. (The actual data is located in ROM.)

`.dtors`: Pointer arrays to global class destructor functions. (The actual data is located in ROM.)

`.gcc_except_table`: Table data for exception handling. (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 that 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(.gcc_except_table)`" means that a section will have its actual data located at an address following another section (in this case, the `.gcc_except_table` 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

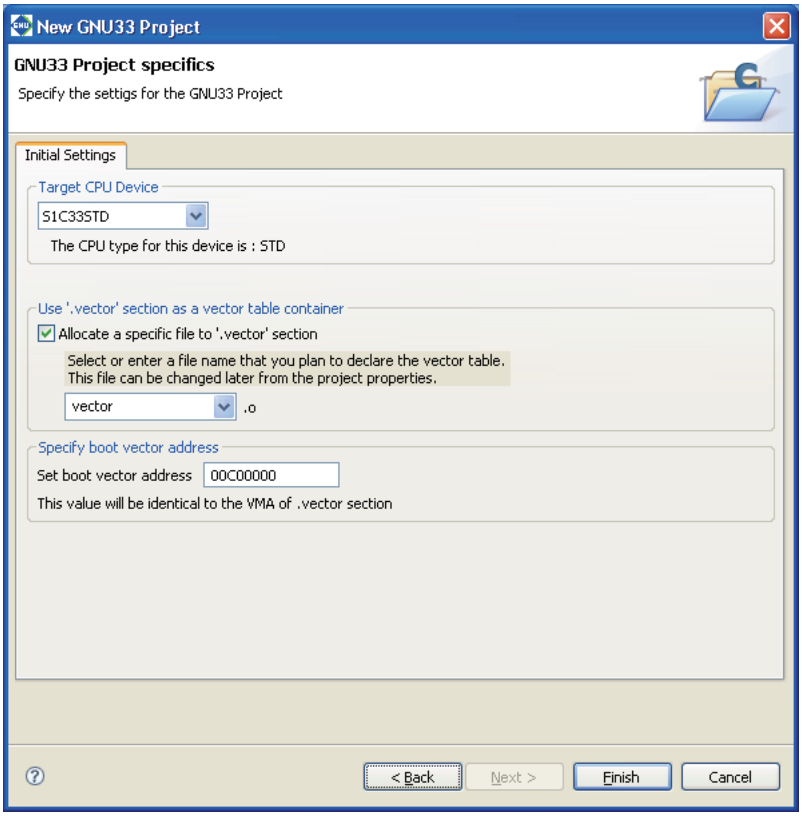
About the `.vector` section

The `.vector` section is an exclusive section provided by the **IDE** to ensure that vector tables will always be located beginning with the vector table base 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 same page of a wizard that you used to select the target CPU when creating a new project. For more information on this wizard, refer to Section 5.4.2, "Creating a New Project".



GNU33 Project specifics

Specify the settigs for the GNU33 Project

Target CPU Device: S1C33STD

The CPU type for this device is : STD

Use '.vector' section as a vector table container

☑ Allocate a specific file to '.vector' section

Select or enter a file name that you plan to declare the vector table.
This file can be changed later from the project properties.

vector .o

Specify boot vector address

Set boot vector address 00C00000

This value will be identical to the VMA of .vector section

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 "00C00000" when "S1C33STD" or "S1C33PE" is selected as the target CPU or "20000000" when "S1C33401" is selected. 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**. It will also 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/C++ sources (`vector.c` or `vector.cpp`)

Declare a vector table with `const` to specify that it be located in the `.rodata` section.

Example:

```
const unsigned long vector[] = {
    (unsigned long)boot,            // 0    0
    0,                              // 4    1
    0,                              // 8    2
                          :
    (unsigned long)dummy,           // 280  70
    (unsigned long)dummy            // 284  71
};
```

For assembler sources (`boot.s`)

Declare a `.rodata` section and write a vector table following it.

Example:

```
.section .rodata, "a"
.long   BOOT                ; 0    0
.long   0,                  ; 4    1
.long   0,                  ; 8    2
               :
.long   DUMMY               ; 280  70
.long   DUMMY               ; 284  71
```

If you are using an existing 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 [GNU33 Linker Script Settings] page.

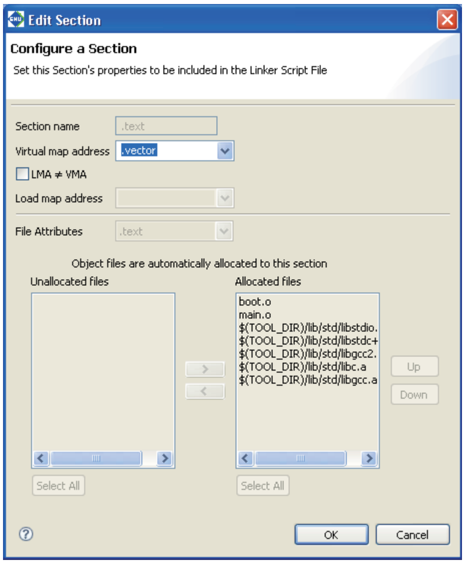
(2) Click the [Edit] button.
This displays the [Edit Section] dialog box.

(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



Edit Section

Configure a Section

Set this Section's properties to be included in the Linker Script File

Section name: .text

Virtual map address: .vector

☐ LMA ≠ VMA

Load map address:

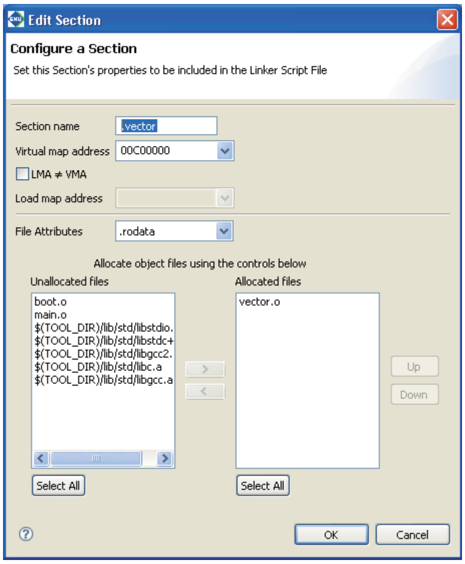
File Attributes: .text

Object files are automatically allocated to this section

Unallocated files:

Allocated files:
boot.o
main.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

User section



Edit Section

Configure a Section

Set this Section's properties to be included in the Linker Script File

Section name: .vector

Virtual map address: 00C00000

☐ LMA ≠ VMA

Load map address:

File Attributes: .rodata

Allocate object files using the controls below

Unallocated files:
boot.o
main.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

Allocated files:
vector.o

Use the upper part of the dialog box to set the location information on a section in an execution file.

[Section name]

Set a 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 eight 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 eight 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 that 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]

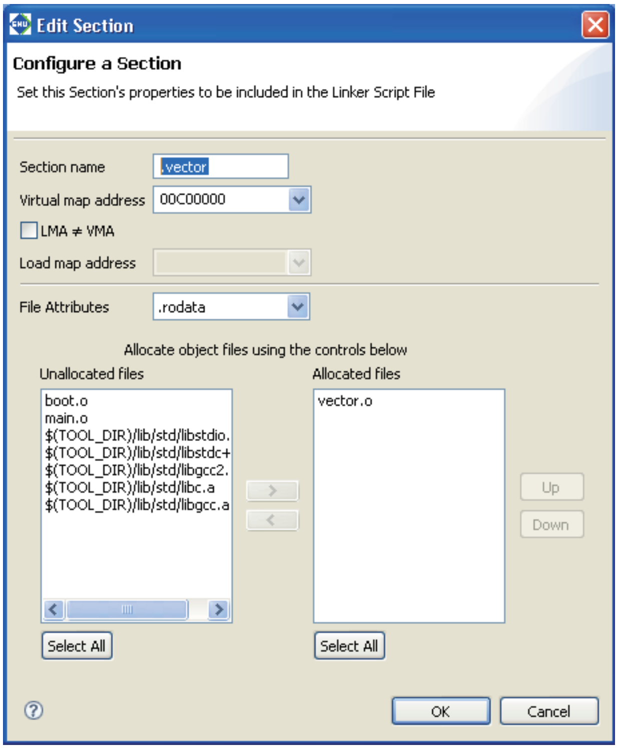
Selects all entries in the respective lists. This button is ineffective for standard sections.

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 [GNU33 Linker Script Settings] page for project properties, select the `.vector` section and click the [Edit] button.
This displays the [Edit Section] dialog box.



Configure a Section

Set this Section's properties to be included in the Linker Script File

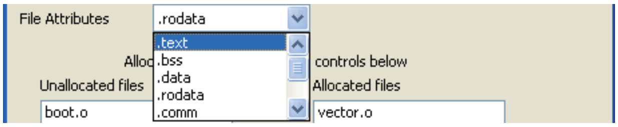
Section name: .vector
Virtual map address: 00C00000
☐ LMA ≠ VMA
Load map address:
File Attributes: .rodata

Allocate object files using the controls below

Unallocated files:
boot.o
main.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

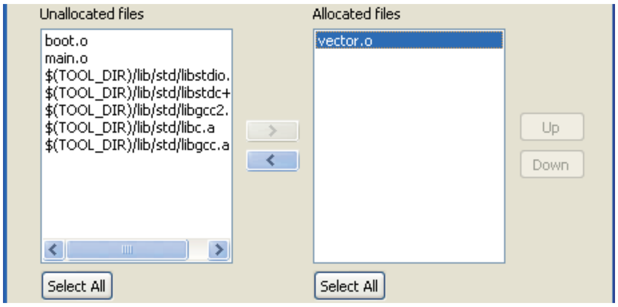
Allocated files:
vector.o

(2) Select `.text` from the [File Attributes] pull-down list.



File Attributes: .rodata
.text
.bss
.data
.rodata
.comm

(3) Select `vector.o` from the selected file list and click the [<] button.

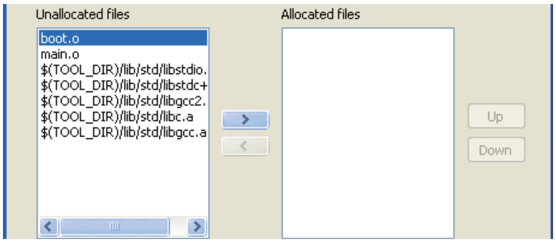


Unallocated files:
boot.o
main.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

Allocated files:
vector.o

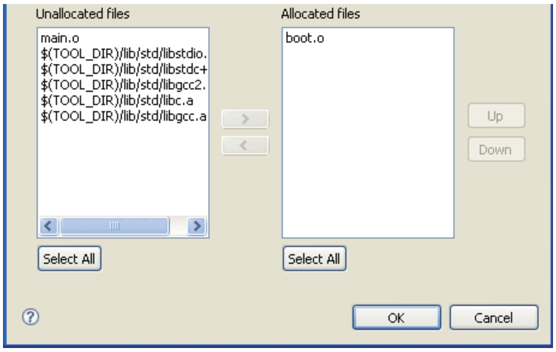
Since no source files are available for `vector.o`, it is removed without being moved to the unselected file list. This operation alone also removes `vector.o` from other section information.

(4) Select `boot.o` from the unselected file list and click the [>] button.


Unallocated files
boot.o
main.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

Allocated files

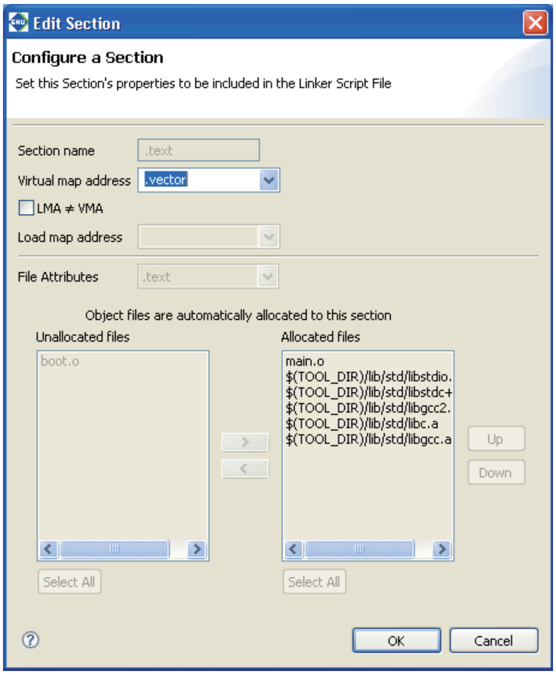
The selected `boot.o` is moved to the selected file list.


Unallocated files
main.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

Allocated files
boot.o

(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.)


Edit Section

Configure a Section

Set this Section's properties to be included in the Linker Script File

Section name: .text
Virtual map address: .vector
☐ LMA ≠ VMA
Load map address:
File Attributes: .text

Object files are automatically allocated to this section

Unallocated files
boot.o

Allocated files
main.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

(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:

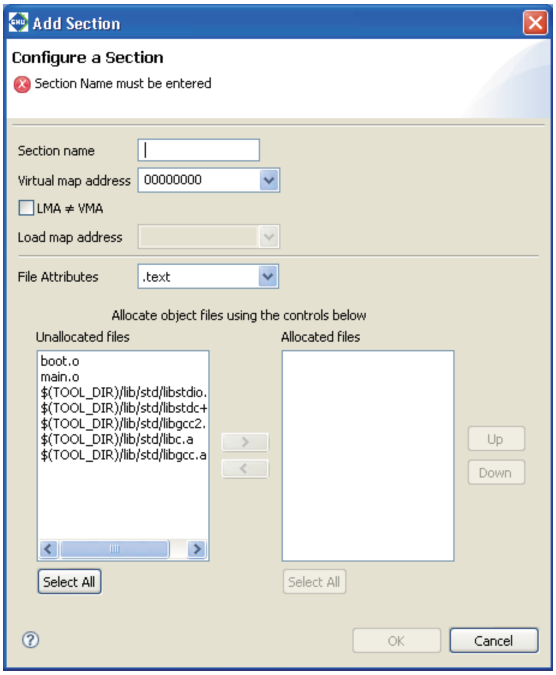
1. The section attribute ([File Attributes]) cannot be changed.
2. The file list can only be referenced; it cannot be manipulated.
3. Object files are added in ascending alphabetical order.

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.



Add Section

Configure a Section

Section Name must be entered

Section name:

Virtual map address: 00000000

☐ LMA ≠ VMA

Load map address:

File Attributes: .text

Allocate object files using the controls below

Unallocated files:
boot.o
main.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

Allocated files:

(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 [GNU33 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] = 00000000)
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] = 00000000)
Section3 ([Virtual map address] = Section1)
Section4 ([Virtual map address] = Section3)

If Section1 is removed, the location address of Section2 changes to (0x)00000000. If this results in problems, reedit the section information.
Section2 ([Virtual map address] = 00000000)
Section3 ([Virtual map address] = Section2)
Section4 ([Virtual map address] = Section3)

Setting the data pointer

The [Default data area pointer] text box of the [GNU33 Linker Script Settings] page is used to set the default data area pointer (address) to be written in a linker script. The data area pointer is provided to reduce the number of instructions by using one of the internal registers of the C33 Core as a pointer and accessing data within the address range confined by the base address stored in that register.
With default settings, a data pointer definition where the base address is designated as address 0 is written in a linker script.

```
/* data pointer symbols */
__dp = 0x00000000;
```

Note: By default, the `-medda32` options for the compiler and assembler build options are set, which disables the data pointer feature. In order to utilize the data pointer, you will have to set the above options disabled. This also requires writing an instruction in the source to initialize the data pointer. For more information, refer to Section 3.7.1, "Data Area".

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 eight 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 GNU33 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

For the sake of simplicity, it is assumed here that the object files do not contain `.comm`, `.ctors`, `.dtors`, and `.gcc_except_table` sections. (A linker script is also created for these sections.)
It is also 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 internal RAM and one external ROM. The external ROM is mapped to memory from address 0xc00000, thereby locating program and data to this memory, as shown in Figure 5.7.7.2. 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 ROM. The internal RAM space starting from address 0x0 has a variable area without initial values that begins with the start address. Subsequent areas are used for variables with initial values. The initial values of variables are stored in ROM and copied from ROM into RAM by an application.

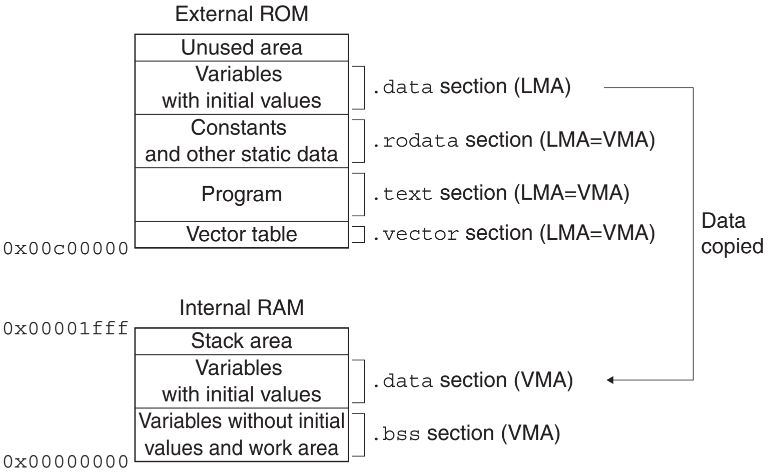

External ROM
Unused area
Variables with initial values
Constants and other static data
Program
Vector table
0x00c00000
.data section (LMA)
.rodata section (LMA=VMA)
.text section (LMA=VMA)
.vector section (LMA=VMA)
Data copied
Internal RAM
0x00001fff
Stack area
Variables with initial values
Variables without initial values and work area
0x00000000
.data section (VMA)
.bss section (VMA)

Figure 5.7.7.2 Example of a memory configuration 1

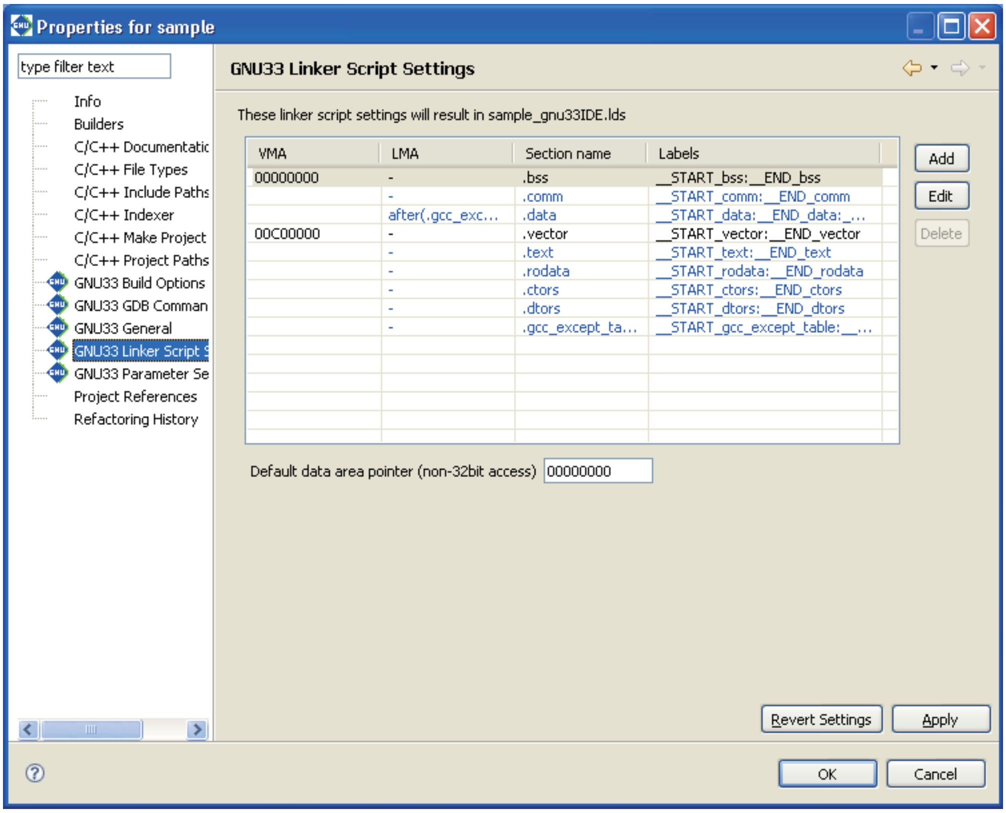
Using default linker script files is the simplest method for using memory in this way.
This section location can be realized without adding to or correcting settings in the [GNU33 Linker Script Settings] dialog box.

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 (0xc00000 and beyond).

Section configuration (contents set in the [GNU33 Linker Script Settings] dialog box)


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
Add
Edit
Delete
Revert Settings
Apply
OK
Cancel

GNU33 Linker Script Settings

These linker script settings will result in sample_gnu33IDE.lds

VMA	LMA	Section name	Labels
00000000	-	.bss	__START_bss:__END_bss
	-	.comm	__START_comm:__END_comm
	after(.gcc_exc...	.data	__START_data:__END_data:_...
00C00000	-	.vector	__START_vector:__END_vector
	-	.text	__START_text:__END_text
	-	.rodata	__START_rodata:__END_rodata
	-	.ctors	__START_ctors:__END_ctors
	-	.dtors	__START_dtors:__END_dtors
	-	.gcc_except_ta...	__START_gcc_except_table:__...

Default data area pointer (non-32bit access) 00000000

These are the contents set by default.
The VMAs (execution addresses) are set so that the `.bss` section is located from address 0x00000000 (start of RAM) and the `.vector` section (`.rodata` section of `boot.o`) is located from address 0x00c00000 (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 `.gcc_except_table` section. In this example, wherein no C++ source are used, the `.ctors`, `.dtors`, and `.gcc_except_table` sections are not significant, but not deleted.
The memory map configuration in Figure 5.7.7.2 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 Gnu33 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c33", "elf32-c33", "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);

SECTIONS
{
```

```
/* location counter */
. = 0x0;

/* section information */
.bss 0x00000000 :
  {
   __START_bss = . ;
   boot.o(.bss)
   main.o(.bss)
   C:/EPSON/gnu33/lib/std/libstdio.a(.bss)
   C:/EPSON/gnu33/lib/std/libstdc++.a(.bss)
   C:/EPSON/gnu33/lib/std/libgcc2.a(.bss)
   C:/EPSON/gnu33/lib/std/libc.a(.bss)
   C:/EPSON/gnu33/lib/std/libgcc.a(.bss)
  }
  __END_bss = . ;

.comm __END_bss :
  {
   __START_comm = . ;
   boot.o(.comm)
   main.o(.comm)
   C:/EPSON/gnu33/lib/std/libstdio.a(.comm)
   C:/EPSON/gnu33/lib/std/libstdc++.a(.comm)
   C:/EPSON/gnu33/lib/std/libgcc2.a(.comm)
   C:/EPSON/gnu33/lib/std/libc.a(.comm)
   C:/EPSON/gnu33/lib/std/libgcc.a(.comm)
  }
  __END_comm = . ;

.data __END_comm : AT( __END_gcc_except_table )
  {
   __START_data = . ;
   boot.o(.data)
   main.o(.data)
   C:/EPSON/gnu33/lib/std/libstdio.a(.data)
   C:/EPSON/gnu33/lib/std/libstdc++.a(.data)
   C:/EPSON/gnu33/lib/std/libgcc2.a(.data)
   C:/EPSON/gnu33/lib/std/libc.a(.data)
   C:/EPSON/gnu33/lib/std/libgcc.a(.data)
  }
  __END_data = . ;

.vector 0x00C00000 :
  {
   __START_vector = . ;
   boot.o(.rodata)
  }
  __END_vector = . ;

.text __END_vector :
  {
   __START_text = . ;
   boot.o(.text)
   main.o(.text)
   C:/EPSON/gnu33/lib/std/libstdio.a(.text)
   C:/EPSON/gnu33/lib/std/libstdc++.a(.text)
   C:/EPSON/gnu33/lib/std/libgcc2.a(.text)
   C:/EPSON/gnu33/lib/std/libc.a(.text)
   C:/EPSON/gnu33/lib/std/libgcc.a(.text)
  }
  __END_text = . ;

.rodata __END_text :
  {
   __START_rodata = . ;
   main.o(.rodata)
   C:/EPSON/gnu33/lib/std/libstdio.a(.rodata)
   C:/EPSON/gnu33/lib/std/libstdc++.a(.rodata)
   C:/EPSON/gnu33/lib/std/libgcc2.a(.rodata)
   C:/EPSON/gnu33/lib/std/libc.a(.rodata)
   C:/EPSON/gnu33/lib/std/libgcc.a(.rodata)
```

```
    }
    __END_rodata = . ;

  .ctors __END_rodata :
    {
     . = ALIGN(4);
     __START_ctors = . ;
     boot.o(.ctors)
     main.o(.ctors)
     C:/EPSON/gnu33/lib/std/libstdio.a(.ctors)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.ctors)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.ctors)
     C:/EPSON/gnu33/lib/std/libc.a(.ctors)
     C:/EPSON/gnu33/lib/std/libgcc.a(.ctors)
    }
    __END_ctors = . ;

  .dtors __END_ctors :
    {
     . = ALIGN(4);
     __START_dtors = . ;
     boot.o(.dtors)
     main.o(.dtors)
     C:/EPSON/gnu33/lib/std/libstdio.a(.dtors)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.dtors)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.dtors)
     C:/EPSON/gnu33/lib/std/libc.a(.dtors)
     C:/EPSON/gnu33/lib/std/libgcc.a(.dtors)
    }
    __END_dtors = . ;

  .gcc_except_table __END_dtors :
    {
     __START_gcc_except_table = . ;
     boot.o(.gcc_except_table)
     main.o(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libstdio.a(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libc.a(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libgcc.a(.gcc_except_table)
    }
    __END_gcc_except_table = . ;

  /* 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.

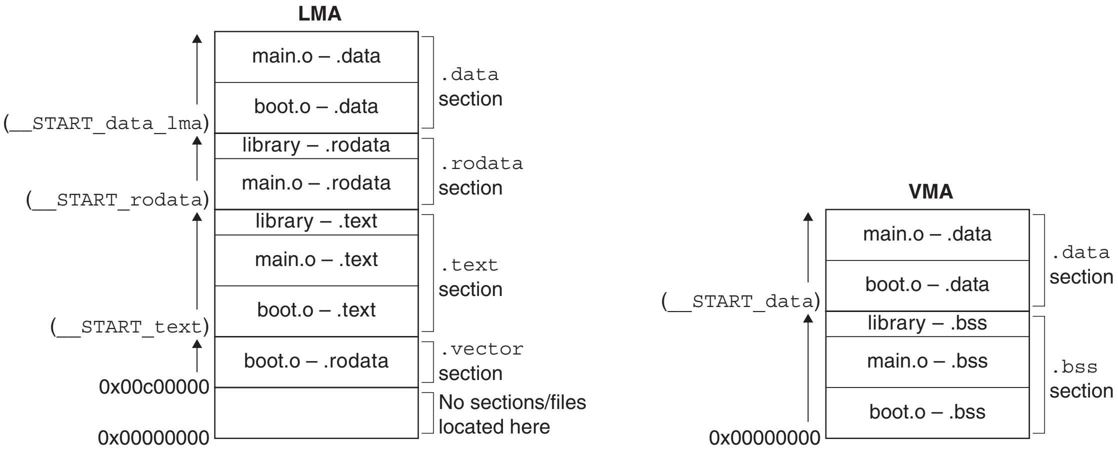

LMA
main.o – .data
boot.o – .data
.data section
(__START_data_lma)
library – .rodata
main.o – .rodata
.rodata section
(__START_rodata)
library – .text
main.o – .text
.text section
boot.o – .text
(__START_text)
boot.o – .rodata
.vector section
0x00c00000
No sections/files located here
0x00000000
VMA
main.o – .data
boot.o – .data
.data section
(__START_data)
library – .bss
main.o – .bss
.bss section
boot.o – .bss
0x00000000

Figure 5.7.7.3 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.

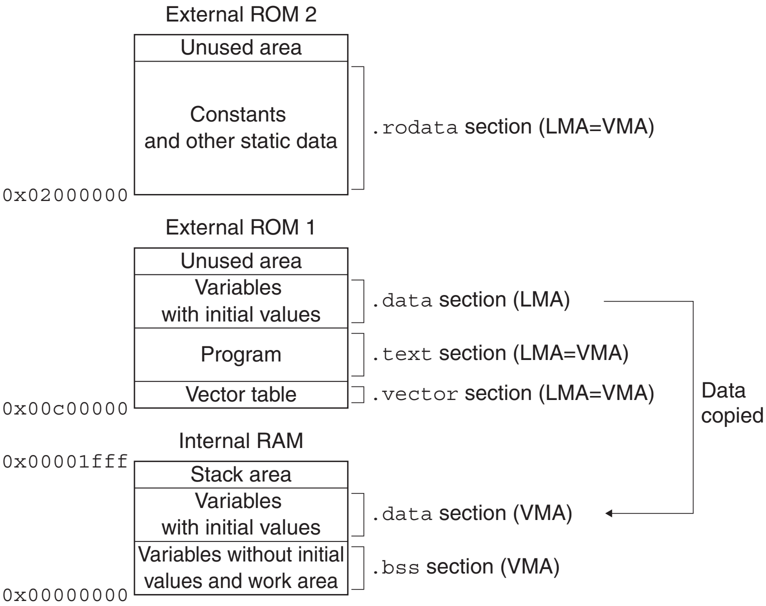

External ROM 2
Unused area
Constants and other static data
.rodata section (LMA=VMA)
0x02000000
External ROM 1
Unused area
Variables with initial values
.data section (LMA)
Program
.text section (LMA=VMA)
Vector table
.vector section (LMA=VMA)
0x00c00000
Data copied
Internal RAM
0x00001fff
Stack area
Variables with initial values
.data section (VMA)
Variables without initial values and work area
.bss section (VMA)
0x00000000

Figure 5.7.7.4 Example of a memory configuration 2

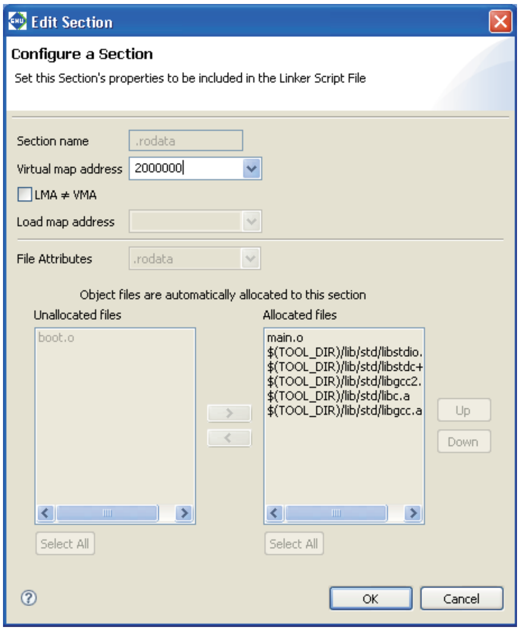
Example of a source file configuration

`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 `.ctors` 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 0x02000000.

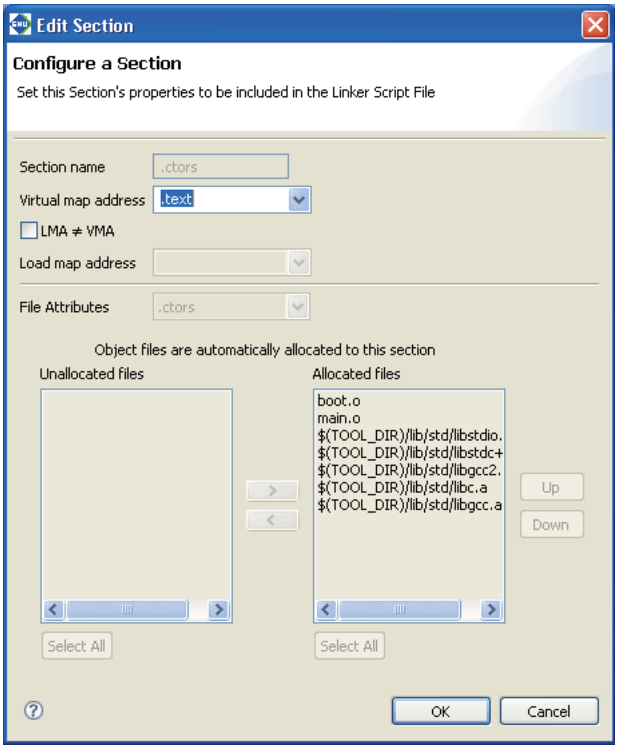

Edit Section
Configure a Section
Set this Section's properties to be included in the Linker Script File
Object files are automatically allocated to this section
Unallocated files
Allocated files
>
<
Up
Down
Select All
Select All
OK
Cancel

Section name .rodata
Virtual map address 2000000
☐ LMA ≠ VMA
Load map address
File Attributes .rodata

Unallocated files:
boot.o

Allocated files:
main.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

2. Correcting the `.ctors` section
   Correct [Virtual map address] to "`.text`".



Edit Section

Configure a Section

Set this Section's properties to be included in the Linker Script File

Section name: .ctors

Virtual map address: .text

☐ LMA ≠ VMA

Load map address:

File Attributes: .ctors

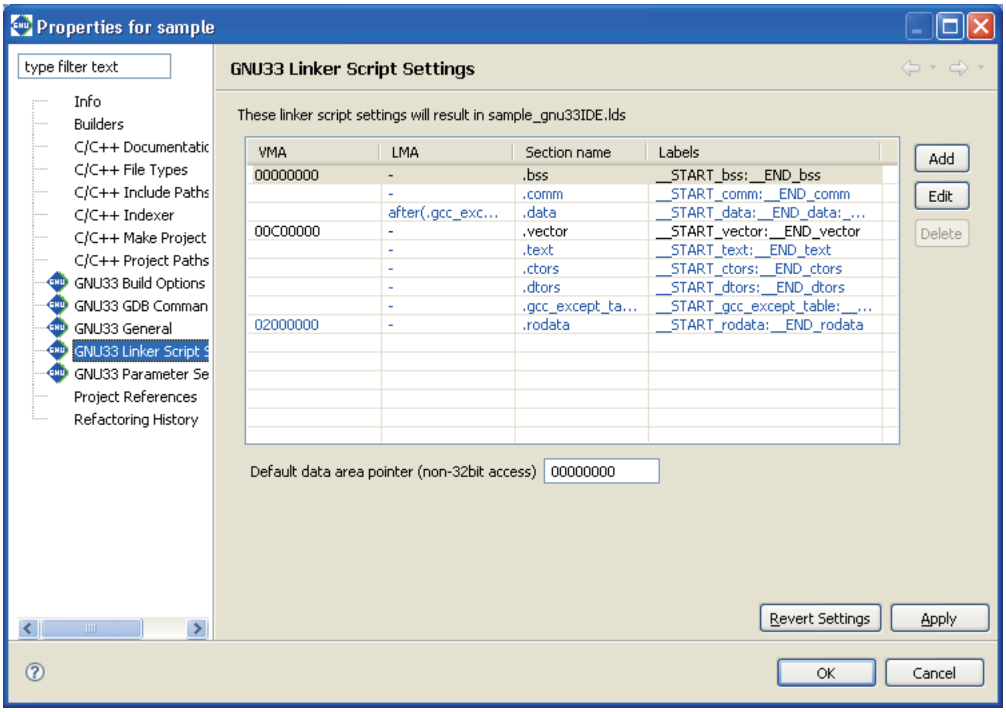
Object files are automatically allocated to this section

Unallocated files

Allocated files

boot.o
main.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

Section configuration (contents set in the [GNU33 Linker Script Settings] dialog box)



Properties for sample

type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History

GNU33 Linker Script Settings

These linker script settings will result in sample_gnu33IDE.lds

VMA	LMA	Section name	Labels
00000000	-	.bss	__START_bss:__END_bss
	-	.comm	__START_comm:__END_comm
	after(.gcc_exc...	.data	__START_data:__END_data:_...
00C00000	-	.vector	__START_vector:__END_vector
	-	.text	__START_text:__END_text
	-	.ctors	__START_ctors:__END_ctors
	-	.dtors	__START_dtors:__END_dtors
	-	.gcc_except_ta...	__START_gcc_except_table:__...
02000000	-	.rodata	__START_rodata:__END_rodata

Default data area pointer (non-32bit access): 00000000

Revert Settings

Linker script and section location

The linker script is generated as shown below.

```
/* Linker Script file generated by Gnu33 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c33", "elf32-c33", "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);

SECTIONS
{
  /* data pointer symbols */
  __dp = 0x00000000;

  /* location counter */
  . = 0x0;

  /* section information */
  .bss 0x00000000 :
    {
     __START_bss = . ;
     boot.o(.bss)
     main.o(.bss)
     C:/EPSON/gnu33/lib/std/libstdio.a(.bss)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.bss)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.bss)
     C:/EPSON/gnu33/lib/std/libc.a(.bss)
     C:/EPSON/gnu33/lib/std/libgcc.a(.bss)
    }
    __END_bss = . ;

  .comm __END_bss :
    {
     __START_comm = . ;
     boot.o(.comm)
     main.o(.comm)
     C:/EPSON/gnu33/lib/std/libstdio.a(.comm)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.comm)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.comm)
     C:/EPSON/gnu33/lib/std/libc.a(.comm)
     C:/EPSON/gnu33/lib/std/libgcc.a(.comm)
    }
    __END_comm = . ;

  .data __END_comm : AT( __END_gcc_except_table )
    {
     __START_data = . ;
     boot.o(.data)
     main.o(.data)
     C:/EPSON/gnu33/lib/std/libstdio.a(.data)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.data)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.data)
     C:/EPSON/gnu33/lib/std/libc.a(.data)
     C:/EPSON/gnu33/lib/std/libgcc.a(.data)
    }
    __END_data = . ;

  .vector 0x00C00000 :
    {
     __START_vector = . ;
     boot.o(.rodata)
    }
    __END_vector = . ;

  .text __END_vector :
    {
     __START_text = . ;
     boot.o(.text)
     main.o(.text)
     C:/EPSON/gnu33/lib/std/libstdio.a(.text)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.text)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.text)
     C:/EPSON/gnu33/lib/std/libc.a(.text)
     C:/EPSON/gnu33/lib/std/libgcc.a(.text)
    }
    __END_text = . ;

  .ctors __END_text :
    {
     . = ALIGN(4);
     __START_ctors = . ;
     boot.o(.ctors)
     main.o(.ctors)
     C:/EPSON/gnu33/lib/std/libstdio.a(.ctors)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.ctors)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.ctors)
     C:/EPSON/gnu33/lib/std/libc.a(.ctors)
     C:/EPSON/gnu33/lib/std/libgcc.a(.ctors)
    }
    __END_ctors = . ;

  .dtors __END_ctors :
    {
     . = ALIGN(4);
     __START_dtors = . ;
     boot.o(.dtors)
     main.o(.dtors)
     C:/EPSON/gnu33/lib/std/libstdio.a(.dtors)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.dtors)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.dtors)
     C:/EPSON/gnu33/lib/std/libc.a(.dtors)
     C:/EPSON/gnu33/lib/std/libgcc.a(.dtors)
    }
    __END_dtors = . ;

  .gcc_except_table __END_dtors :
    {
     __START_gcc_except_table = . ;
     boot.o(.gcc_except_table)
     main.o(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libstdio.a(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libc.a(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libgcc.a(.gcc_except_table)
    }
    __END_gcc_except_table = . ;

  .rodata 0x02000000 :
    {
     __START_rodata = . ;
     main.o(.rodata)
     C:/EPSON/gnu33/lib/std/libstdio.a(.rodata)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.rodata)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.rodata)
     C:/EPSON/gnu33/lib/std/libc.a(.rodata)
     C:/EPSON/gnu33/lib/std/libgcc.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.

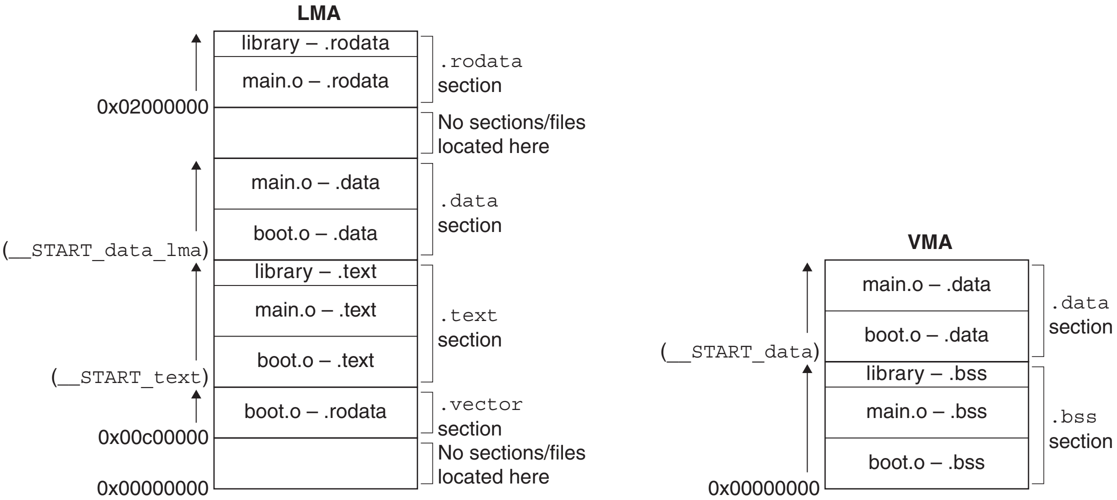

LMA
library – .rodata
main.o – .rodata
.rodata section
0x02000000
No sections/files located here
main.o – .data
boot.o – .data
.data section
(__START_data_lma)
library – .text
main.o – .text
boot.o – .text
.text section
(__START_text)
boot.o – .rodata
.vector section
0x00c00000
No sections/files located here
0x00000000
VMA
main.o – .data
boot.o – .data
.data section
(__START_data)
library – .bss
main.o – .bss
boot.o – .bss
.bss section
0x00000000

Figure 5.7.7.5 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 5.7.7.6, 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.

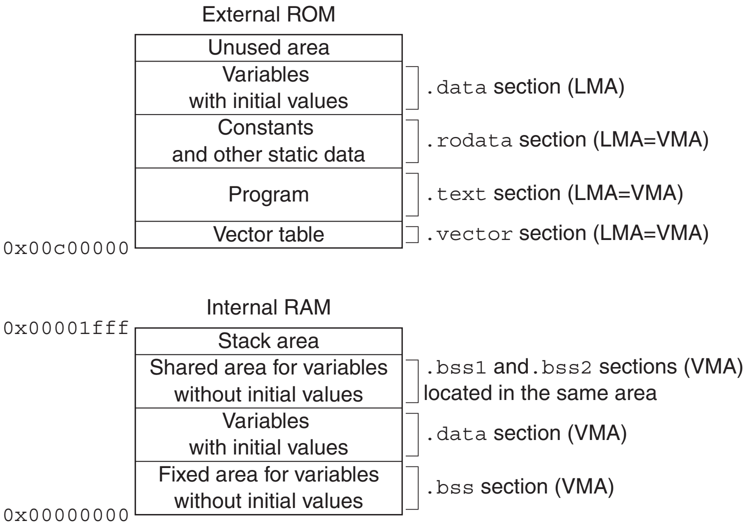

External ROM
Unused area
Variables with initial values
.data section (LMA)
Constants and other static data
.rodata section (LMA=VMA)
Program
.text section (LMA=VMA)
Vector table
.vector section (LMA=VMA)
0x00c00000
Internal RAM
0x00001fff
Stack area
Shared area for variables without initial values
.bss1 and .bss2 sections (VMA) located in the same area
Variables with initial values
.data section (VMA)
Fixed area for variables without initial values
.bss section (VMA)
0x00000000

Figure 5.7.7.6 Example for shared data area

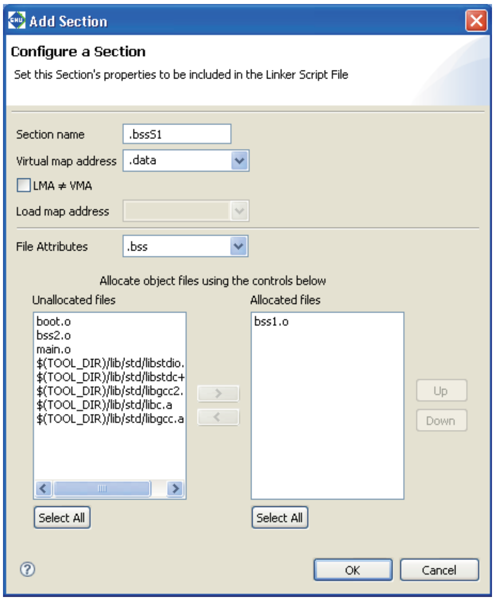
Example of a source file configuration

`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.


Add Section

Configure a Section

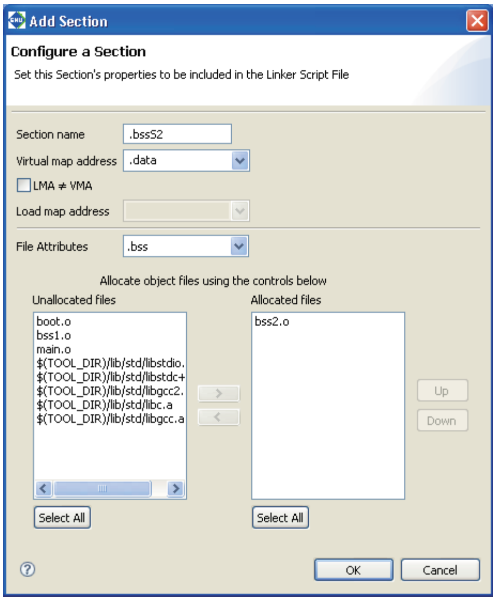
Set this Section's properties to be included in the Linker Script File

Section name: .bssS1
Virtual map address: .data
☐ LMA ≠ VMA
Load map address:
File Attributes: .bss

Allocate object files using the controls below

Unallocated files:
boot.o
bss2.o
main.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

Allocated files:
bss1.o


Add Section

Configure a Section

Set this Section's properties to be included in the Linker Script File

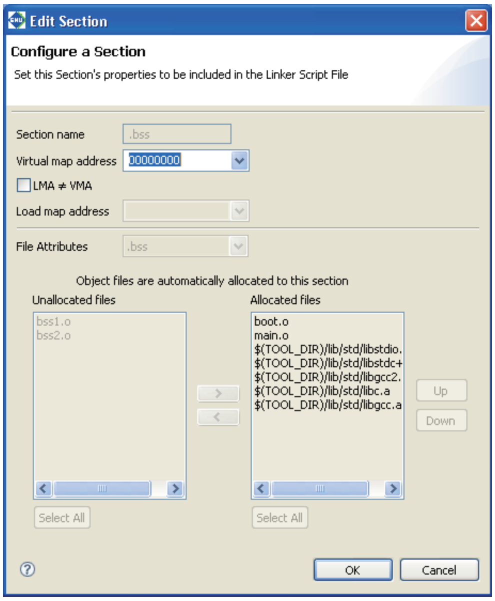
Section name: .bssS2
Virtual map address: .data
☐ LMA ≠ VMA
Load map address:
File Attributes: .bss

Allocate object files using the controls below

Unallocated files:
boot.o
bss1.o
main.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

Allocated files:
bss2.o

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.


Edit Section

Configure a Section

Set this Section's properties to be included in the Linker Script File

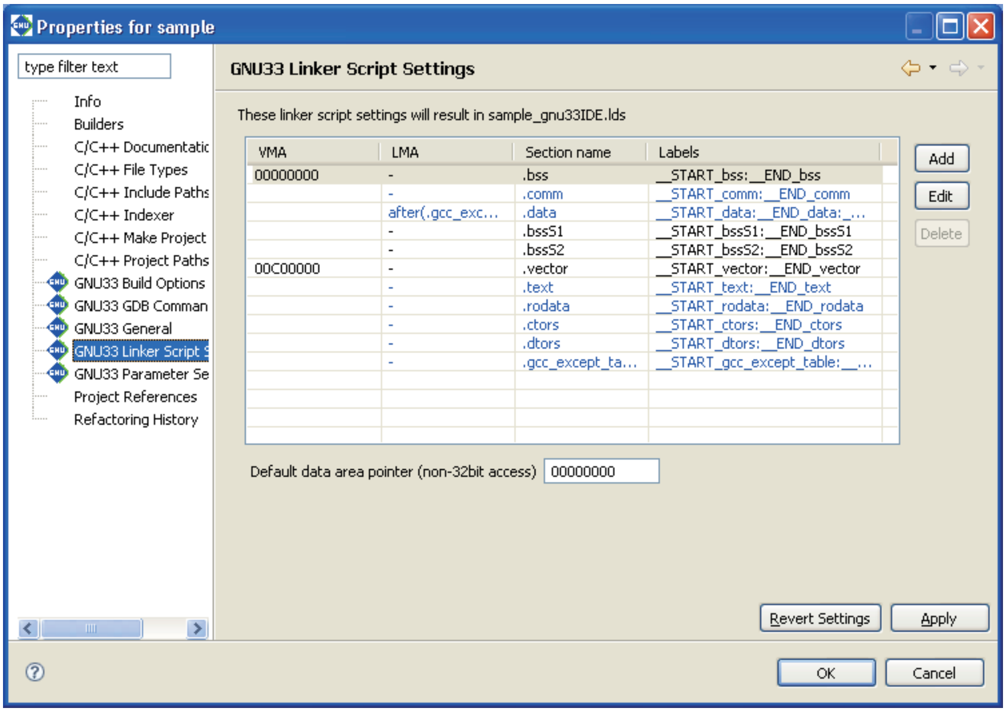
Section name: .bss
Virtual map address: 00000000
☐ LMA ≠ VMA
Load map address:
File Attributes: .bss

Object files are automatically allocated to this section

Unallocated files:
bss1.o
bss2.o

Allocated files:
boot.o
main.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

Section configuration (content set in the [GNU33 Linker Script Settings] dialog box)


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Linker Script Settings
Add
Edit
Delete
Revert Settings
Apply
OK
Cancel

These linker script settings will result in sample_gnu33IDE.lds

VMA	LMA	Section name	Labels
00000000	-	.bss	__START_bss:__END_bss
	-	.comm	__START_comm:__END_comm
	after(.gcc_exc...	.data	__START_data:__END_data:_...
	-	.bssS1	__START_bssS1:__END_bssS1
	-	.bssS2	__START_bssS2:__END_bssS2
00C00000	-	.vector	__START_vector:__END_vector
	-	.text	__START_text:__END_text
	-	.rodata	__START_rodata:__END_rodata
	-	.ctors	__START_ctors:__END_ctors
	-	.dtors	__START_dtors:__END_dtors
	-	.gcc_except_ta...	__START_gcc_except_table:__...

Default data area pointer (non-32bit access) 00000000

Linker script and section location

The linker script is generated as shown below.

```
/* Linker Script file generated by Gnu33 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c33", "elf32-c33", "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);

SECTIONS
{
  /* data pointer symbols */
  __dp = 0x00000000;

  /* location counter */
  . = 0x0;

  /* section information */
  .bss 0x00000000 :
    {
     __START_bss = . ;
     boot.o(.bss)
     main.o(.bss)
     C:/EPSON/gnu33/lib/std/libstdio.a(.bss)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.bss)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.bss)
     C:/EPSON/gnu33/lib/std/libc.a(.bss)
     C:/EPSON/gnu33/lib/std/libgcc.a(.bss)
    }
    __END_bss = . ;

  .comm __END_bss :
    {
     __START_comm = . ;
     boot.o(.comm)
     bss1.o(.comm)
     bss2.o(.comm)
     main.o(.comm)
     C:/EPSON/gnu33/lib/std/libstdio.a(.comm)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.comm)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.comm)
     C:/EPSON/gnu33/lib/std/libc.a(.comm)
     C:/EPSON/gnu33/lib/std/libgcc.a(.comm)
    }
    __END_comm = . ;

  .data __END_comm : AT( __END_gcc_except_table )
    {
     __START_data = . ;
     boot.o(.data)
     bss1.o(.data)
     bss2.o(.data)
     main.o(.data)
     C:/EPSON/gnu33/lib/std/libstdio.a(.data)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.data)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.data)
     C:/EPSON/gnu33/lib/std/libc.a(.data)
     C:/EPSON/gnu33/lib/std/libgcc.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 0x00C00000 :
    {
     __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/gnu33/lib/std/libstdio.a(.text)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.text)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.text)
     C:/EPSON/gnu33/lib/std/libc.a(.text)
     C:/EPSON/gnu33/lib/std/libgcc.a(.text)
    }
    __END_text = . ;

  .rodata __END_text :
    {
     __START_rodata = . ;
     bss1.o(.rodata)
     bss2.o(.rodata)
     main.o(.rodata)
     C:/EPSON/gnu33/lib/std/libstdio.a(.rodata)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.rodata)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.rodata)
     C:/EPSON/gnu33/lib/std/libc.a(.rodata)
     C:/EPSON/gnu33/lib/std/libgcc.a(.rodata)
    }
    __END_rodata = . ;

  .ctors __END_rodata :
    {
     . = ALIGN(4);
     __START_ctors = . ;
     boot.o(.ctors)
     bss1.o(.ctors)
     bss2.o(.ctors)
     main.o(.ctors)
     C:/EPSON/gnu33/lib/std/libstdio.a(.ctors)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.ctors)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.ctors)
     C:/EPSON/gnu33/lib/std/libc.a(.ctors)
     C:/EPSON/gnu33/lib/std/libgcc.a(.ctors)
    }
    __END_ctors = . ;
  .dtors __END_ctors :
    {
     . = ALIGN(4);
     __START_dtors = . ;
     boot.o(.dtors)
     bss1.o(.dtors)
     bss2.o(.dtors)
     main.o(.dtors)
     C:/EPSON/gnu33/lib/std/libstdio.a(.dtors)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.dtors)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.dtors)
     C:/EPSON/gnu33/lib/std/libc.a(.dtors)
     C:/EPSON/gnu33/lib/std/libgcc.a(.dtors)
    }
    __END_dtors = . ;
  .gcc_except_table __END_dtors :
    {
     __START_gcc_except_table = . ;
     boot.o(.gcc_except_table)
     bss1.o(.gcc_except_table)
     bss2.o(.gcc_except_table)
     main.o(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libstdio.a(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libc.a(.gcc_except_table)
     C:/EPSON/gnu33/lib/std/libgcc.a(.gcc_except_table)
    }
    __END_gcc_except_table = . ;
  /* load address symbols */
  __START_data_lma = LOADADDR( .data );
  __END_data_lma = __START_data_lma + SIZEOF( .data );
}
```

Shown below are section locations and file configurations.

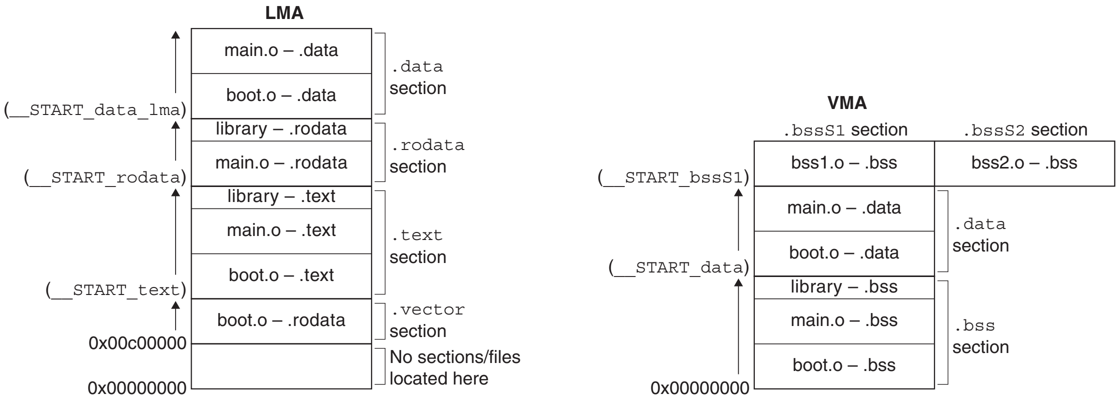

LMA
main.o – .data
boot.o – .data
.data section
(__START_data_lma)
library – .rodata
main.o – .rodata
.rodata section
(__START_rodata)
library – .text
main.o – .text
boot.o – .text
.text section
(__START_text)
boot.o – .rodata
.vector section
0x00c00000
No sections/files located here
0x00000000
VMA
.bssS1 section
.bssS2 section
(__START_bssS1)
bss1.o – .bss
bss2.o – .bss
main.o – .data
boot.o – .data
.data section
(__START_data)
library – .bss
main.o – .bss
boot.o – .bss
.bss section
0x00000000

Figure 5.7.7.7 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.

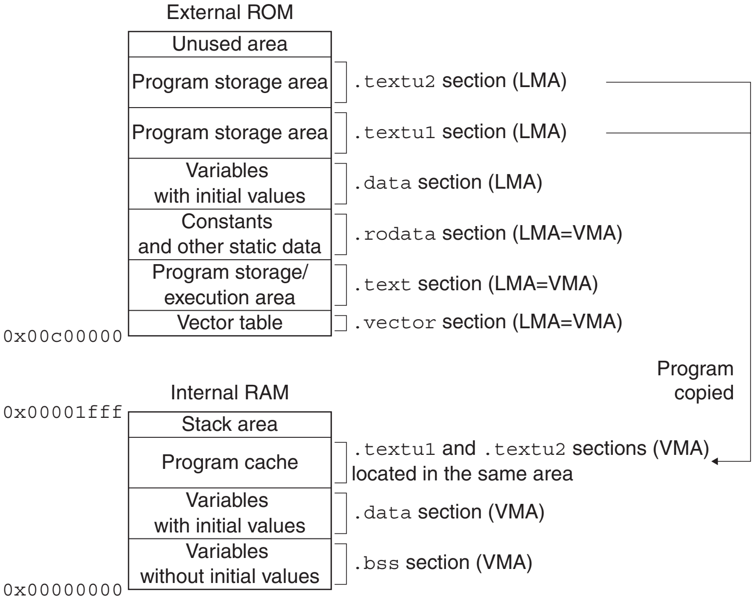

External ROM
Unused area
Program storage area
Program storage area
Variables with initial values
Constants and other static data
Program storage/ execution area
Vector table
0x00c00000
.textu2 section (LMA)
.textu1 section (LMA)
.data section (LMA)
.rodata section (LMA=VMA)
.text section (LMA=VMA)
.vector section (LMA=VMA)
Program copied
Internal RAM
0x00001fff
Stack area
Program cache
Variables with initial values
Variables without initial values
0x00000000
.textu1 and .textu2 sections (VMA) located in the same area
.data section (VMA)
.bss section (VMA)

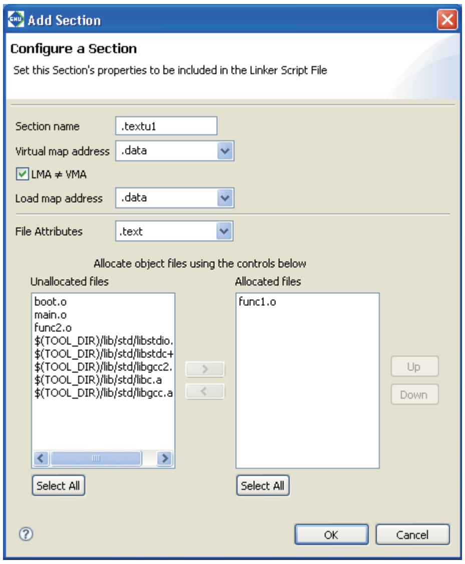
Figure 5.7.7.8 Allocating a storage area for the program in RAM

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.



Add Section

Configure a Section

Set this Section's properties to be included in the Linker Script File

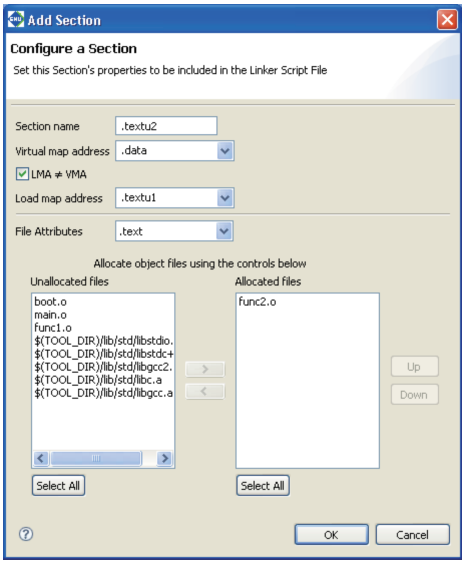
Section name: .textu1
Virtual map address: .data
☑ LMA ≠ VMA
Load map address: .data
File Attributes: .text

Allocate object files using the controls below

Unallocated files:
boot.o
main.o
func2.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

Allocated files:
func1.o

>
<
Up
Down
Select All
Select All
OK
Cancel



Add Section

Configure a Section

Set this Section's properties to be included in the Linker Script File

Section name: .textu2
Virtual map address: .data
☑ LMA ≠ VMA
Load map address: .textu1
File Attributes: .text

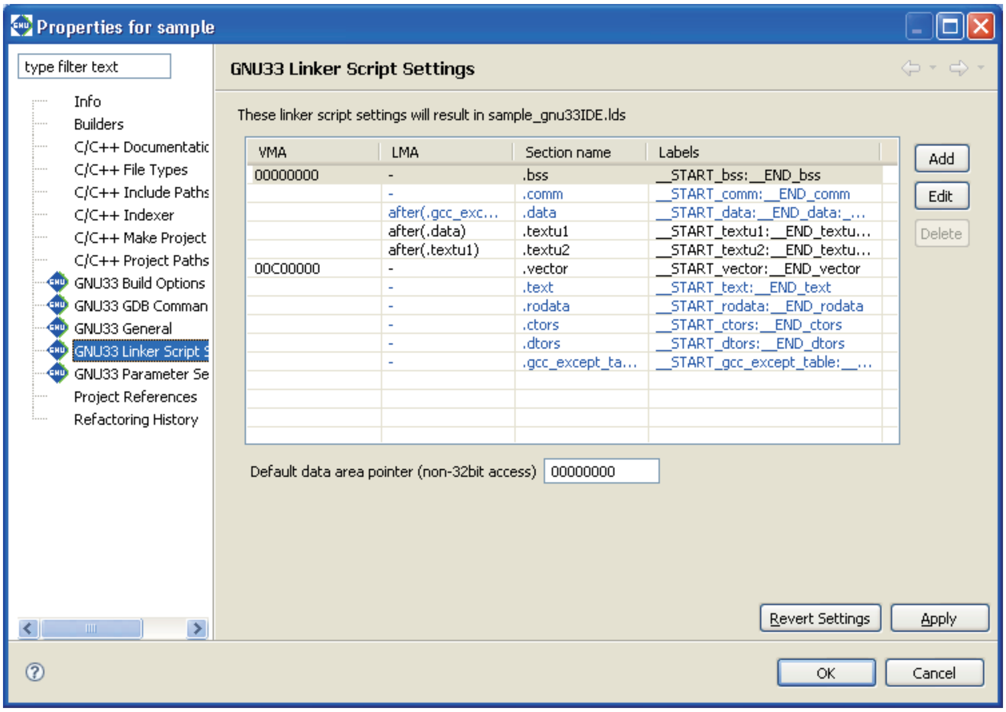
Allocate object files using the controls below

Unallocated files:
boot.o
main.o
func1.o
$(TOOL_DIR)/lib/std/libstdio.
$(TOOL_DIR)/lib/std/libstdc+
$(TOOL_DIR)/lib/std/libgcc2.
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a

Allocated files:
func2.o

>
<
Up
Down
Select All
Select All
OK
Cancel

Section configuration (content set in the [GNU33 Linker Script Settings] dialog box)


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Linker Script Settings
Add
Edit
Delete
Revert Settings
Apply
OK
Cancel

These linker script settings will result in sample_gnu33IDE.lds

VMA	LMA	Section name	Labels
00000000	-	.bss	__START_bss:__END_bss
	-	.comm	__START_comm:__END_comm
	after(.gcc_exc...	.data	__START_data:__END_data:_...
	after(.data)	.textu1	__START_textu1:__END_textu...
	after(.textu1)	.textu2	__START_textu2:__END_textu...
00C00000	-	.vector	__START_vector:__END_vector
	-	.text	__START_text:__END_text
	-	.rodata	__START_rodata:__END_rodata
	-	.ctors	__START_ctors:__END_ctors
	-	.dtors	__START_dtors:__END_dtors
	-	.gcc_except_ta...	__START_gcc_except_table:__...

Default data area pointer (non-32bit access) 00000000

Linker script and section location

The linker script is generated, as shown below.

```
/* Linker Script file generated by Gnu33 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c33", "elf32-c33", "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);

SECTIONS
{
  /* data pointer symbols */
  __dp = 0x00000000;

  /* location counter */
  . = 0x0;

  /* section information */
  .bss 0x00000000 :
    {
     __START_bss = . ;
     boot.o(.bss)
     func1.o(.bss)
     func2.o(.bss)
     main.o(.bss)
     C:/EPSON/gnu33/lib/std/libstdio.a(.bss)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.bss)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.bss)
     C:/EPSON/gnu33/lib/std/libc.a(.bss)
     C:/EPSON/gnu33/lib/std/libgcc.a(.bss)
    }
    __END_bss = . ;

  .comm __END_bss :
    {
     __START_comm = . ;
     boot.o(.comm)
     func1.o(.comm)
     func2.o(.comm)
     main.o(.comm)
     C:/EPSON/gnu33/lib/std/libstdio.a(.comm)
```

```
      C:/EPSON/gnu33/lib/std/libstdc++.a(.comm)
      C:/EPSON/gnu33/lib/std/libgcc2.a(.comm)
      C:/EPSON/gnu33/lib/std/libc.a(.comm)
      C:/EPSON/gnu33/lib/std/libgcc.a(.comm)
     }
     __END_comm = . ;

   .data __END_comm : AT( __END_gcc_except_table )
     {
      __START_data = . ;
      boot.o(.data)
      func1.o(.data)
      func2.o(.data)
      main.o(.data)
      C:/EPSON/gnu33/lib/std/libstdio.a(.data)
      C:/EPSON/gnu33/lib/std/libstdc++.a(.data)
      C:/EPSON/gnu33/lib/std/libgcc2.a(.data)
      C:/EPSON/gnu33/lib/std/libc.a(.data)
      C:/EPSON/gnu33/lib/std/libgcc.a(.data)
     }
     __END_data = . ;

   .textu1 __END_data : AT( __START_data_lma + SIZEOF( .data ) )
     {
      __START_textu1 = . ;
      func1.o(.text)
     }
     __END_textu1 = . ;

   .textu2 __END_data : AT( __START_textu1_lma + SIZEOF( .textu1 ) )
     {
      __START_textu2 = . ;
      func2.o(.text)
     }
     __END_textu2 = . ;

   .vector 0x00C00000 :
     {
      __START_vector = . ;
      boot.o(.rodata)
     }
     __END_vector = . ;

   .text __END_vector :
     {
      __START_text = . ;
      boot.o(.text)
      main.o(.text)
      C:/EPSON/gnu33/lib/std/libstdio.a(.text)
      C:/EPSON/gnu33/lib/std/libstdc++.a(.text)
      C:/EPSON/gnu33/lib/std/libgcc2.a(.text)
      C:/EPSON/gnu33/lib/std/libc.a(.text)
      C:/EPSON/gnu33/lib/std/libgcc.a(.text)
     }
     __END_text = . ;

   .rodata __END_text :
     {
      __START_rodata = . ;
      func1.o(.rodata)
      func2.o(.rodata)
      main.o(.rodata)
      C:/EPSON/gnu33/lib/std/libstdio.a(.rodata)
      C:/EPSON/gnu33/lib/std/libstdc++.a(.rodata)
      C:/EPSON/gnu33/lib/std/libgcc2.a(.rodata)
      C:/EPSON/gnu33/lib/std/libc.a(.rodata)
      C:/EPSON/gnu33/lib/std/libgcc.a(.rodata)
     }
     __END_rodata = . ;
```

```
    .ctors __END_rodata :
      {
       . = ALIGN(4);
       __START_ctors = . ;
       boot.o(.ctors)
       func1.o(.ctors)
       func2.o(.ctors)
       main.o(.ctors)
       C:/EPSON/gnu33/lib/std/libstdio.a(.ctors)
       C:/EPSON/gnu33/lib/std/libstdc++.a(.ctors)
       C:/EPSON/gnu33/lib/std/libgcc2.a(.ctors)
       C:/EPSON/gnu33/lib/std/libc.a(.ctors)
       C:/EPSON/gnu33/lib/std/libgcc.a(.ctors)
      }
      __END_ctors = . ;

    .dtors __END_ctors :
      {
       . = ALIGN(4);
       __START_dtors = . ;
       boot.o(.dtors)
       func1.o(.dtors)
       func2.o(.dtors)
       main.o(.dtors)
       C:/EPSON/gnu33/lib/std/libstdio.a(.dtors)
       C:/EPSON/gnu33/lib/std/libstdc++.a(.dtors)
       C:/EPSON/gnu33/lib/std/libgcc2.a(.dtors)
       C:/EPSON/gnu33/lib/std/libc.a(.dtors)
       C:/EPSON/gnu33/lib/std/libgcc.a(.dtors)
      }
      __END_dtors = . ;

    .gcc_except_table __END_dtors :
      {
       __START_gcc_except_table = . ;
       boot.o(.gcc_except_table)
       func1.o(.gcc_except_table)
       func2.o(.gcc_except_table)
       main.o(.gcc_except_table)
       C:/EPSON/gnu33/lib/std/libstdio.a(.gcc_except_table)
       C:/EPSON/gnu33/lib/std/libstdc++.a(.gcc_except_table)
       C:/EPSON/gnu33/lib/std/libgcc2.a(.gcc_except_table)
       C:/EPSON/gnu33/lib/std/libc.a(.gcc_except_table)
       C:/EPSON/gnu33/lib/std/libgcc.a(.gcc_except_table)
      }
      __END_gcc_except_table = . ;

    /* 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.

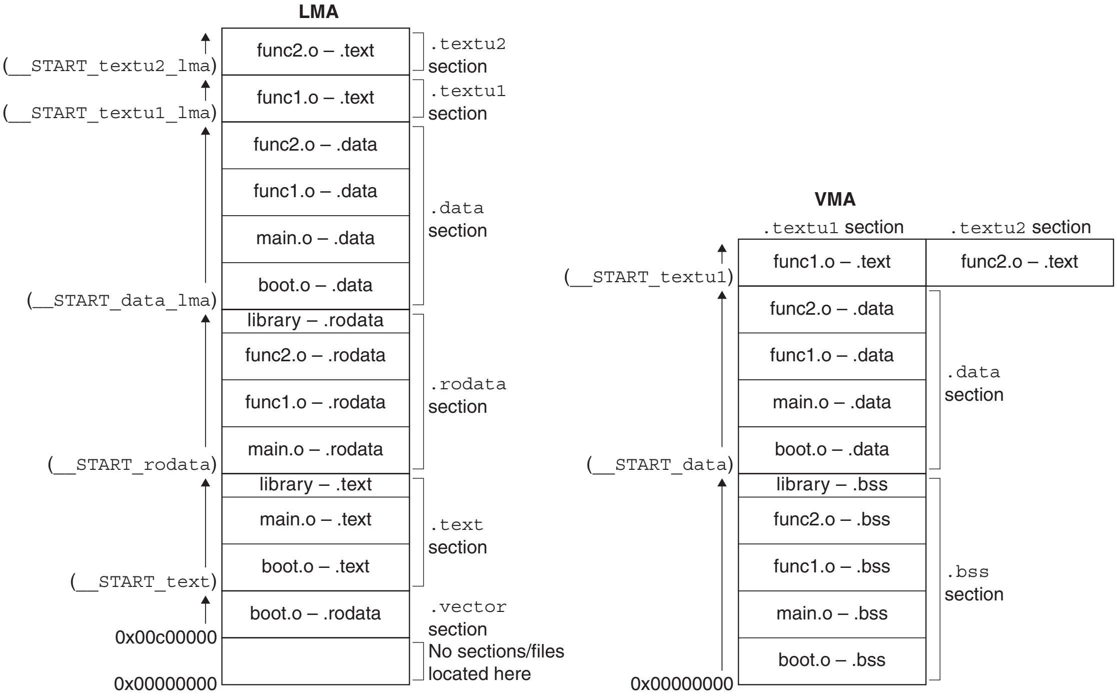

LMA
func2.o – .text
.textu2 section
(__START_textu2_lma)
func1.o – .text
.textu1 section
(__START_textu1_lma)
func2.o – .data
func1.o – .data
main.o – .data
boot.o – .data
.data section
(__START_data_lma)
library – .rodata
func2.o – .rodata
func1.o – .rodata
main.o – .rodata
.rodata section
(__START_rodata)
library – .text
main.o – .text
boot.o – .text
.text section
(__START_text)
boot.o – .rodata
.vector section
0x00c00000
No sections/files located here
0x00000000
VMA
.textu1 section
.textu2 section
func1.o – .text
func2.o – .text
(__START_textu1)
func2.o – .data
func1.o – .data
main.o – .data
boot.o – .data
.data section
(__START_data)
library – .bss
func2.o – .bss
func1.o – .bss
main.o – .bss
boot.o – .bss
.bss section
0x00000000

Figure 5.7.7.9 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 `.rodata` section with (for example) no LMA specified will result in positioning the `.rodata` section behind the end VMA address of the `.textu2` section. Do not locate any VMA-only sections behind LMA ≠ VMA sections.

5.7.8 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*>`_gnu33IDE.mak`) and dependency file (<*source name*>`.d`)
   - Linker script file (<*project name*>`_gnu33IDE.lds`)
   - Parameter file (<*project name*>`_gnu33IDE.par`)*
   - Command file (<*project name*>`_gnu33IDE.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*>`_gnu33IDE.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 [GNU33 File Builder] currently selected check box in the [Builders] page of the [Properties] dialog box.

About object files generated from a link source

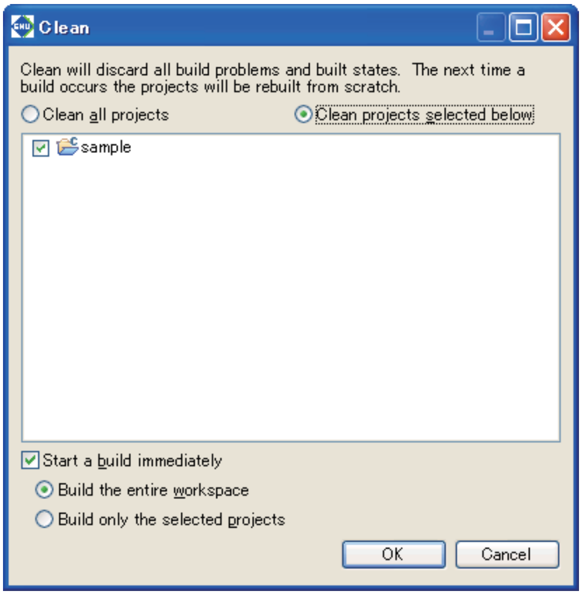
The object files for linked source files located outside the project folder will be generated directly under the project folder.

5.7.9 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.



Clean will discard all build problems and built states. The next time a build occurs the projects will be rebuilt from scratch.

○ Clean all projects ● Clean projects selected below

☑ sample

☑ Start a build immediately

● Build the entire workspace

○ Build only the selected projects

(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.

Select [Build the entire workspace] to build all projects in the workspace.

Select [Build only the selected projects] to build only the projects selected in the above step.

(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).

- When a build fails due to file privileges when a build is executed immediately after importing a project.

5.7.10 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.

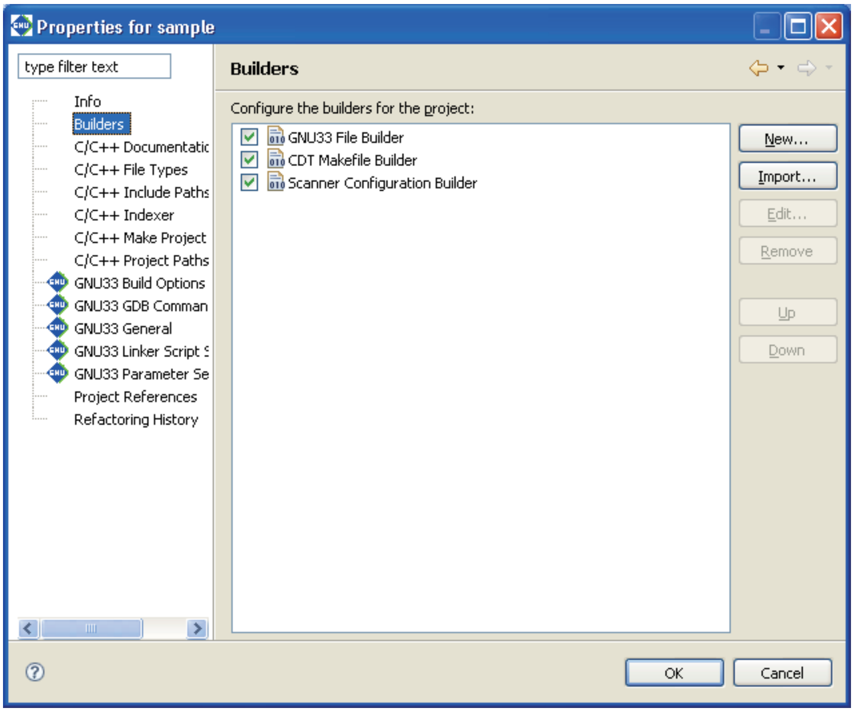
(1) In the [C/C++ Projects] or [Navigator] view, select the project you want to build using the makefile you created.

(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*>`_gnu33IDE.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.

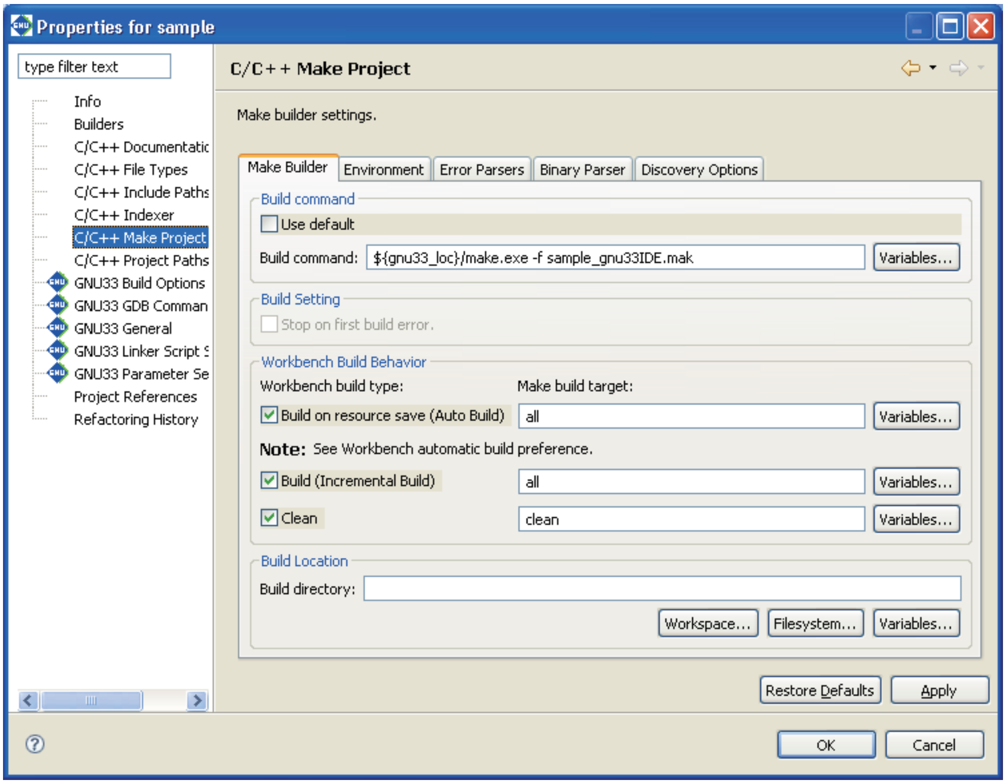

Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
Builders
Configure the builders for the project:
☑ GNU33 File Builder
☑ CDT Makefile Builder
☑ Scanner Configuration Builder
New...
Import...
Edit...
Remove
Up
Down
OK
Cancel

(5) Deselect the [GNU33 File Builder] check box.
The following files will no longer be generated during a build process:

- Makefile (<*project name*>`_gnu33IDE.mak`)
- Linker script file (<*project name*>`_gnu33IDE.lds`)
- Parameter file (<*project name*>`_gnu33IDE.par`)
- Command file (<*project name*>`_gnu33IDE.cmd`)

Selecting the [GNU33 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 [GNU33 File Builder] check box selected. Or select the [GNU33 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.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
C/C++ Make Project
Make Builder
Environment
Error Parsers
Binary Parser
Discovery Options
Build command
Build Setting
Workbench Build Behavior
Build Location
Variables...
Workspace...
Filesystem...
Restore Defaults
Apply
OK
Cancel

Make builder settings.

☐ Use default

Build command: ${gnu33_loc}/make.exe -f sample_gnu33IDE.mak

☐ Stop on first build error.

Workbench build type: Make build target:

☑ Build on resource save (Auto Build) all

Note: See Workbench automatic build preference.

☑ Build (Incremental Build) all

☑ Clean clean

Build directory:

(7) Deselect the [Use default] check box and correct the [Build command:] command line.

Example: Change to the makefile named "`user.mak`".

```
${gnu33_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)
- Trace result acquisition (ICD 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>*`_gnu33IDE.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.

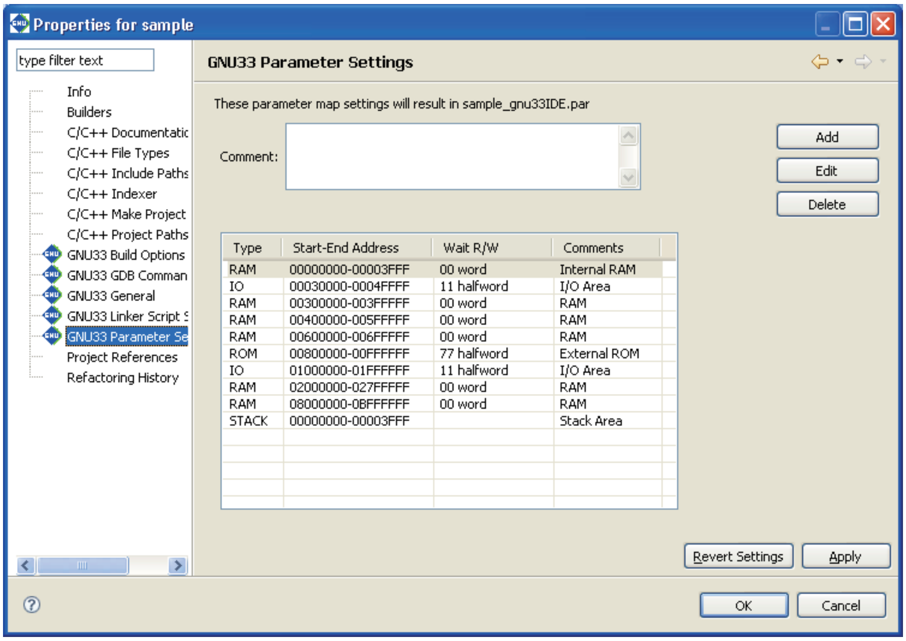
Parameter setup page

Use the [GNU33 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 [GNU33 Parameter Settings] from the properties list.


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Parameter Settings
Add
Edit
Delete
Revert Settings
Apply
OK
Cancel

These parameter map settings will result in sample_gnu33IDE.par

Comment:

Type	Start-End Address	Wait R/W	Comments
RAM	00000000-00003FFF	00 word	Internal RAM
IO	00030000-0004FFFF	11 halfword	I/O Area
RAM	00300000-003FFFFF	00 word	RAM
RAM	00400000-005FFFFF	00 word	RAM
RAM	00600000-006FFFFF	00 word	RAM
ROM	00800000-00FFFFFF	77 halfword	External ROM
IO	01000000-01FFFFFF	11 halfword	I/O Area
RAM	02000000-027FFFFF	00 word	RAM
RAM	08000000-0BFFFFFF	00 word	RAM
STACK	00000000-00003FFF		Stack Area

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", "half-word", and "word" 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 "S1C33STD" or "S1C33PE" is selected for the CPU type

Area	Start-end address	Wait states (R/W)	Access size
RAM	`0x00000000-0x00003FFF`	0/0	word
IO	`0x00030000-0x0004FFFF`	1/1	halfword
RAM	`0x00300000-0x003FFFFF`	0/0	word
RAM	`0x00400000-0x005FFFFF`	0/0	word
RAM	`0x00600000-0x006FFFFF`	0/0	word
ROM	`0x00800000-0x00FFFFFF`	7/7	halfword
IO	`0x01000000-0x01FFFFFF`	1/1	halfword
RAM	`0x02000000-0x027FFFFF`	0/0	word
RAM	`0x08000000-0x0BFFFFFF`	0/0	word
STACK	`0x00000000-0x00003FFF`	–	–

When "S1C33401" is selected for the CPU type

Area	Start-end address	Wait states (R/W)	Access size
RAM	`0x00000000-0x00007FFF`	0/0	word
IO	`0x00030000-0x0004FFFF`	1/1	halfword
RAM	`0x00200000-0x002FFFFF`	0/0	word
RAM	`0x00300000-0x003FFFFF`	0/0	word
RAM	`0x00600000-0x006FFFFF`	0/0	word
RAM	`0x01000000-0x011FFFFF`	0/0	word
RAM	`0x10000000-0x11FFFFFF`	0/0	word
ROM	`0x20000000-0x200FFFFF`	7/7	halfword
RAM	`0x80000000-0x805FFFFF`	0/0	word
STACK	`0x00000000-0x00007FFF`	–	–

Content of a parameter file

When "S1C33STD" is selected for the CPU type

```
# Parameter file generated by Gnu33 Plug-in for Eclipse

RAM     00000000 00003FFF 00W      # Internal RAM
IO      00030000 0004FFFF 11H      # I/O Area
RAM     00300000 003FFFFF 00W      # RAM
RAM     00400000 005FFFFF 00W      # RAM
RAM     00600000 006FFFFF 00W      # RAM
ROM     00800000 00FFFFFF 77H      # External ROM
IO      01000000 01FFFFFF 11H      # I/O Area
RAM     02000000 027FFFFF 00W      # RAM
RAM     08000000 0BFFFFFF 00W      # RAM
STACK   00000000 00003FFF          # Stack Area
```

When "S1C33PE" is selected for the CPU type

```
# Parameter file generated by Gnu33 Plug-in for Eclipse

C33_PE

RAM     00000000 00003FFF 00W      # Internal RAM
IO      00030000 0004FFFF 11H      # I/O Area
RAM     00300000 003FFFFF 00W      # RAM
RAM     00400000 005FFFFF 00W      # RAM
RAM     00600000 006FFFFF 00W      # RAM
ROM     00800000 00FFFFFF 77H      # External ROM
IO      01000000 01FFFFFF 11H      # I/O Area
RAM     02000000 027FFFFF 00W      # RAM
RAM     08000000 0BFFFFFF 00W      # RAM
STACK   00000000 00003FFF          # Stack Area
```

When "S1C33401" is selected for the CPU type

```
# Parameter file generated by Gnu33 Plug-in for Eclipse

C33_ADVANCED

RAM     00000000 00007FFF 00W      # Internal RAM
IO      00030000 0004FFFF 11H      # I/O Area
RAM     00200000 002FFFFF 00W      # RAM
RAM     00300000 003FFFFF 00W      # RAM
RAM     00600000 006FFFFF 00W      # RAM
RAM     01000000 011FFFFF 00W      # RAM
RAM     10000000 11FFFFFF 00W      # RAM
ROM     20000000 200FFFFF 77H      # External ROM
RAM     80000000 805FFFFF 00W      # RAM
STACK   00000000 00007FFF          # 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 [GNU33 Parameter Settings] page, click to select the area you want to edit.

(2) Click the [Edit] button.
This displays the [Edit Parameter] dialog box.

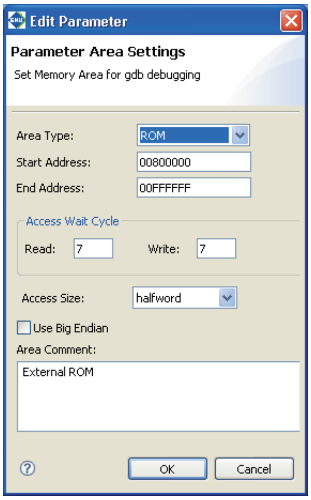
(3) Make the required settings based on the explanation given below. Click [OK].

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



Parameter Area Settings

Set Memory Area for gdb debugging

Area Type: ROM

Start Address: 00800000

End Address: 00FFFFFF

Access Wait Cycle

Read: 7 Write: 7

Access Size: halfword

☐ Use Big Endian

Area Comment:

External ROM

[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 [GNU33 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].

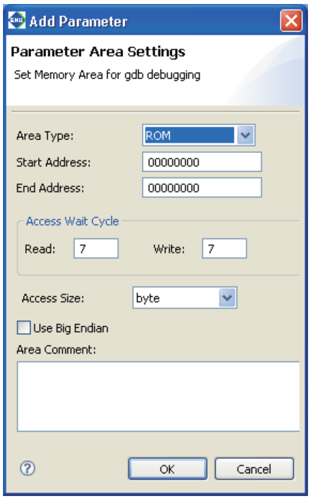
[Cancel]

Discards all modifications and closes the dialog box. The area list in the [GNU33 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.



Parameter Area Settings

Set Memory Area for gdb debugging

Area Type: ROM

Start Address: 00000000

End Address: 00000000

Access Wait Cycle

Read: 7 Write: 7

Access Size: byte

☐ Use Big Endian

Area Comment:

(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 [GNU33 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.

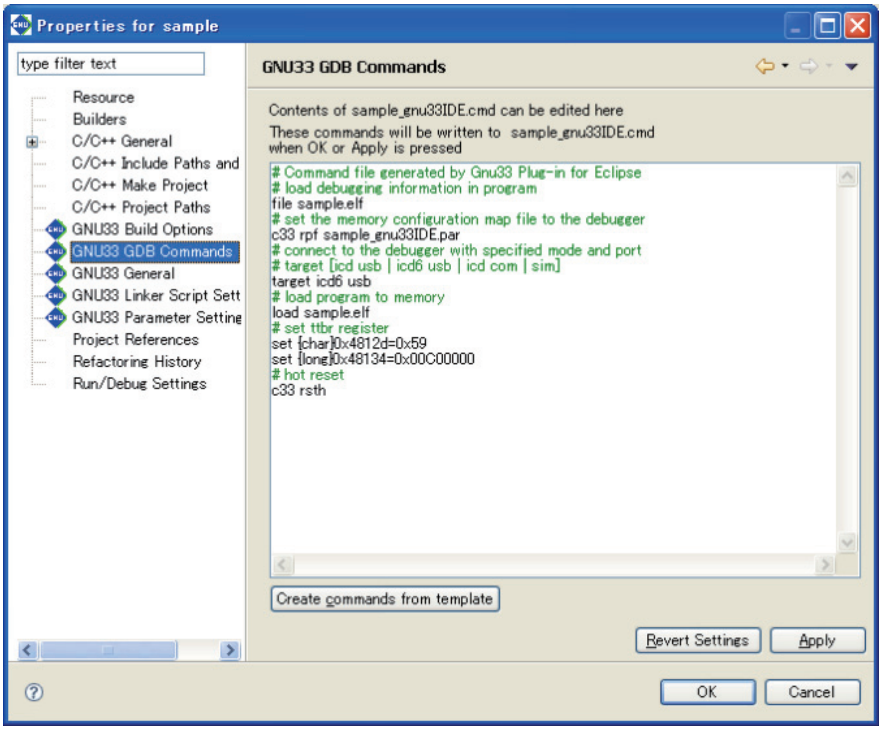
5.8.2 Setting the Debugger Startup Commands

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 [GNU33 GDB Commands] from the properties list.



Properties for sample
type filter text
Resource
Builders
C/C++ General
C/C++ Include Paths and
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Commands
GNU33 General
GNU33 Linker Script Sett
GNU33 Parameter Setting
Project References
Refactoring History
Run/Debug Settings
GNU33 GDB Commands
Create commands from template
Revert Settings
Apply
OK
Cancel

Contents of sample_gnu33IDE.cmd can be edited here

These commands will be written to sample_gnu33IDE.cmd
when OK or Apply is pressed

```
# Command file generated by Gnu33 Plug-in for Eclipse
# load debugging information in program
file sample.elf
# set the memory configuration map file to the debugger
c33 rpf sample_gnu33IDE.par
# connect to the debugger with specified mode and port
# target [icd usb | icd6 usb | icd com | sim]
target icd6 usb
# load program to memory
load sample.elf
# set ttbr register
set {char}0x4812d=0x59
set {long}0x48134=0x00C00000
# hot reset
c33 rsth
```

This page allows the user to directly edit a debugger startup command file. The edit area shows the contents of <*project name*>`_gnu33IDE.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*>`_gnu33IDE.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.

Example: Command file generated by default settings

```
# Command file generated by Gnu33 Plug-in for Eclipse
# load debugging information in program
file sample.elf                                     . . .  Loads debug information.
# set the memory configuration map file to the debugger
c33 rpf sample_gnu33IDE.par                         . . .  Loads a parameter file.
# connect to the debugger with specified mode and port
# target [icd usb | icd6 usb | icd com | sim]
target icd6 usb                                     . . .  Sets connect mode (for ICD Ver6).*1
# load program to memory
load sample.elf                                     . . .  Loads elf object file.
# set ttbr register
set {char}0x4812d=0x59                              . . .  Sets boot vector address (for C33 STD).*2
set {long}0x48134=0x00C00000
# hot reset
c33 rsth                                            . . .  Hot reset
```

*1 The default setting shows the command to set the debugger in ICD Ver6 mode. If another connect 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. The following shows the command lines for connect modes other than ICD Ver3/Ver4:

`target icd comN`	When ICD Ver2 or MON is selected (*N* = COM port number)
`target icd6 usb`	When ICD Ver6 is selected
`target sim`	When Simulator is selected

*2 The command appearing here depends on the target CPU and boot vector address value set in the [GNU33 General] page of the [Properties] dialog box. The boot vector address value may also be changed in the [Create a simple startup command] dialog box shown below. The following shows the boot vector address set commands for S1C33401 and C33 PE (addresses are the default values):

`set $ttbr=0x20000000`	When S1C33401 is selected
`set $ttbr=0x00C00000`	When S1C33PE is selected

This command file is executed when the debugger starts up making the following initial settings:

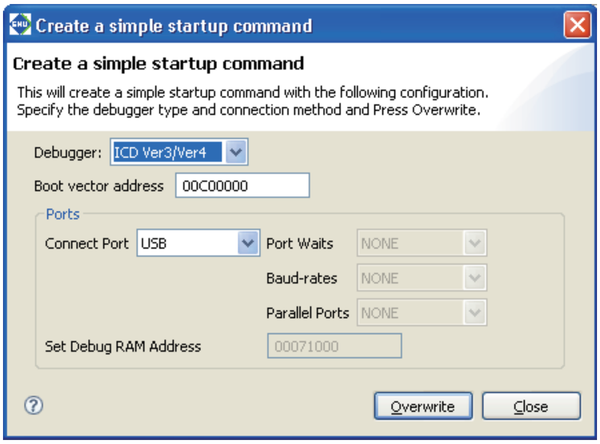
1. Import debug information and parameter map information.
2. Set connect mode.
3. Load the object file to be debugged.
4. Set the PC (program counter) to the boot address.

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 [GNU33 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]



Create a simple startup command

Create a simple startup command

This will create a simple startup command with the following configuration.
Specify the debugger type and connection method and Press Overwrite.

Debugger: ICD Ver3/Ver4

Boot vector address 00C00000

Ports

Connect Port USB

Port Waits NONE

Baud-rates NONE

Parallel Ports NONE

Set Debug RAM Address 00071000

[Debugger:]

Select the debugger (connect mode) to connect to. You will see five choices, but the selectable options will change according to the target CPU selected in the [GNU33 General] page.

ICD Ver6	When using ICD Ver. 6 (S5U1C33001H1400) to debug
ICD Ver3/Ver4	When using ICD Ver. 3 (S5U1C33001H1100) or ICD Ver. 4 (S5U1C33001H1200) to debug
ICD Ver2	When using ICD Ver. 2 (S5U1C33000H) to debug
MON	When using the debug monitor (S5U1C330M1D1 + S5U1C330M2S) to debug
Simulator	To debug with a PC only.

Table 5.8.2.1 Target CPUs and connect modes

Mode	C33 STD Core	C33 PE Core	S1C33401
ICD Ver6	○	○	○
ICD Ver3/Ver4	○	○	○
ICD Ver2	○	×	×
MON	○	×	×
Simulator	○	○	○

[Boot vector address]

Enter the boot vector address in hexadecimal notation (omit 0x). If the CPU type is C33 STD or C33 PE Core, the address must be specified in 400KB increments. For the S1CC33401, the address may be specified in 1KB increments.

The **IDE** generates a command file that includes a command for setting the boot vector address with this value. The value appearing here by default is the address that was specified when the project was created.

[Ports]

Set the desired COM port if you selected ICD Ver2 or MON for [Debugger:] above. If you selected ICD Ver3/Ver.4, ICD Ver6 or Simulator, the port is set to USB and NONE, and no settings are required.

[Connect Port]

Select the COM port (COM1–COM8) to connect with the S5U1C33000H (ICD Ver2) or S5U1C330M1D1 (MON).

[Port Waits]

Specify the duration of the wait state to be inserted when the COM port on the PC side is opened. Set a value ranging from 0 to 20 seconds. Some PCs (such as laptops) may not be able to initiate communications normally unless a wait state is inserted after the COM port is opened.

[Baud-rates]

Select the baud rate for the COM port: 115200 bps or 38400 bps. If you selected MON for [Debugger:], you must select 115200 bps.

[Parallel Ports]

Specify a port number to select the parallel port for transferring programs to the target. Specify NONE, LPT1, or LPT2, which correspond to no parallel ports used, parallel port 1, and parallel port 2, respectively. This selection is enabled only if you selected ICD Ver2 for [Debugger:].

[Set Debug RAM Address]

Set the memory address on the target side to be used by the debugger during a debug process. This setting is available only if you selected S1C33PE for the target CPU in the [GNU33 General] page and ICD Ver3/Ver4 or ICD Ver 6 for [Debugger:] above. The [Set Debug RAM Address] command is valid only for this combination of parameter selections.

Make sure the RAM area specified here is not used in a program.

[Overwrite]

Applies the above settings to the command file shown in the [GNU33 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 [GNU33 GDB Commands] page does not reflect the contents changed in the dialog box.

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

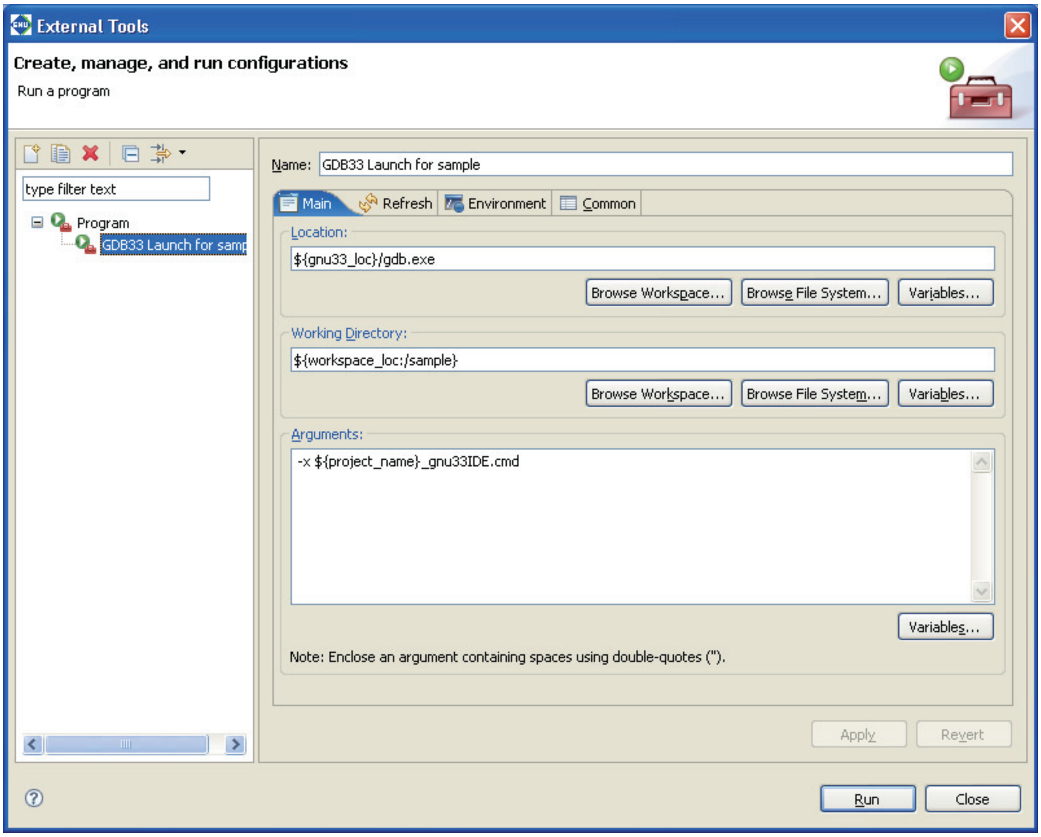
Described below is the procedure for launching the debugger for the first time for the current project.

(1) Do one of the following:

- Select [External Tools] > [External Tools Configurations...] from the [Run] menu.
- Select [External Tools...] from the [External Tools] shortcut in the window toolbar.

This displays the [External Tools Configurations] dialog box.

(2) Select [GDB33 Launch for <*project name*>] from the tree list.



External Tools

Create, manage, and run configurations

Run a program

Name: GDB33 Launch for sample

Location: ${gnu33_loc}/gdb.exe

Working Directory: ${workspace_loc:/sample}

Arguments: -x ${project_name}_gnu33IDE.cmd

Note: Enclose an argument containing spaces using double-quotes (").

type filter text
Program
GDB33 Launch for samp
Main
Refresh
Environment
Common
Browse Workspace...
Browse File System...
Variables...
Browse Workspace...
Browse File System...
Variables...
Variables...
Apply
Revert
Run
Close

∗ ${gnu33_loc} is resolved to the full path for \gnu33 (e.g., C:/EPSON/gnu33) 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 [External Tools Configurations] dialog box, do one of the following to start the second and subsequent debug sessions:

- Select [External Tools] > [GDB33 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 [GDB33 Launch for <*project name*>] from the [External Tools] shortcut in the window toolbar.

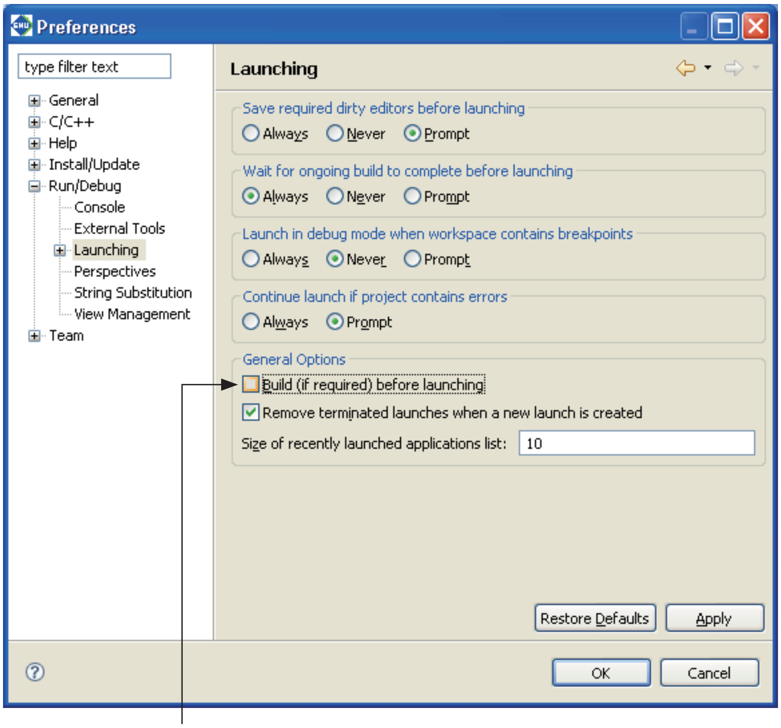
Precautions

- 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.

- Launching the debugger from the [External Tools] menu may perform a build process before the debugger starts up (when a source or other file in the project has been modified). Note that the debugger starts up if an error occurs during the build.
  If you do not want to execute a build before the debugger starts up, disable it by the procedure below.

  (1) Open the [Run/Debug] > [Launching] page after displaying the [Preferences] dialog box by selecting [Preferences...] from the [Window] menu.


Preferences
type filter text
General
C/C++
Help
Install/Update
Run/Debug
Console
External Tools
Launching
Perspectives
String Substitution
View Management
Team
Restore Defaults
Apply
OK
Cancel

Launching

Save required dirty editors before launching
○ Always ○ Never ● Prompt

Wait for ongoing build to complete before launching
● Always ○ Never ○ Prompt

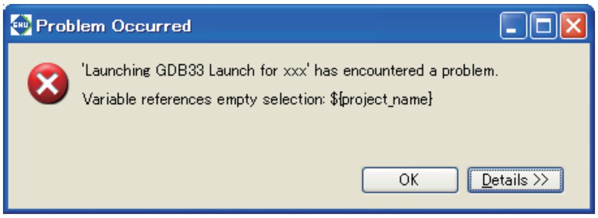
Launch in debug mode when workspace contains breakpoints
○ Always ● Never ○ Prompt

Continue launch if project contains errors
○ Always ● Prompt

General Options
☐ Build (if required) before launching
☑ Remove terminated launches when a new launch is created
Size of recently launched applications list: 10

  (2) Deselect the [Build (if required) before launching] check box.

- If you click the [External Tools] button on the toolbar to launch the debugger, the following dialog box may appear, and the debugger may not launch.



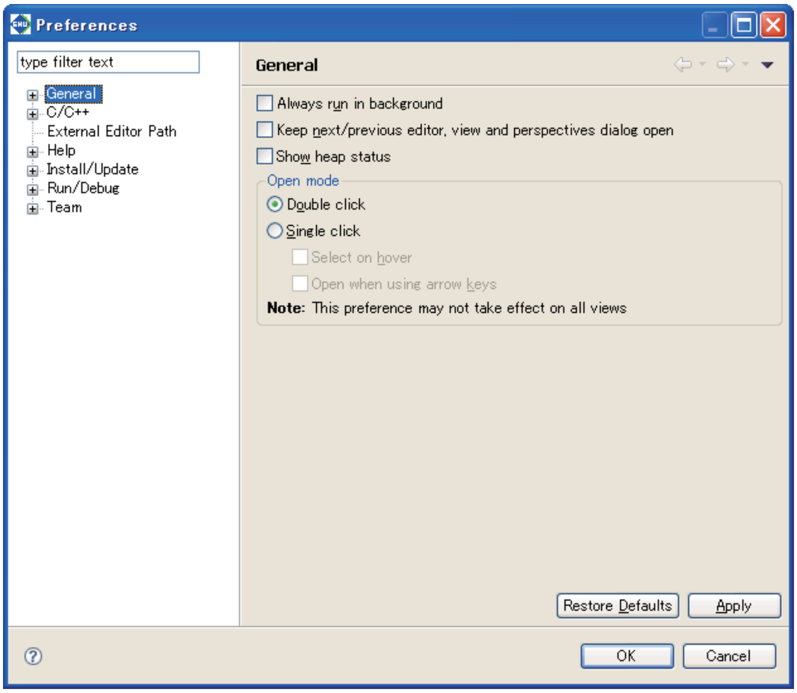
'Launching GDB33 Launch for xxx' has encountered a problem.
Variable references empty selection: ${project_name}

If this happens, select the project icon in the [C/C++ Projects] or [Navigator] view and click the [External Tools] button once again.

Or, change "-x ${program_name}_gnu33IDE.cmd" in [Arguments] in the [Main] tab screen of the [External Tools Configurations] dialog box to "-x <project name>_gnu33IDE.cmd."

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.



☐ Always run in background
☐ Keep next/previous editor, view and perspectives dialog open
☐ Show heap status

Open mode
● Double click
○ Single click
☐ Select on hover
☐ Open when using arrow keys

Note: This preference may not take effect on all views

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 pages 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]

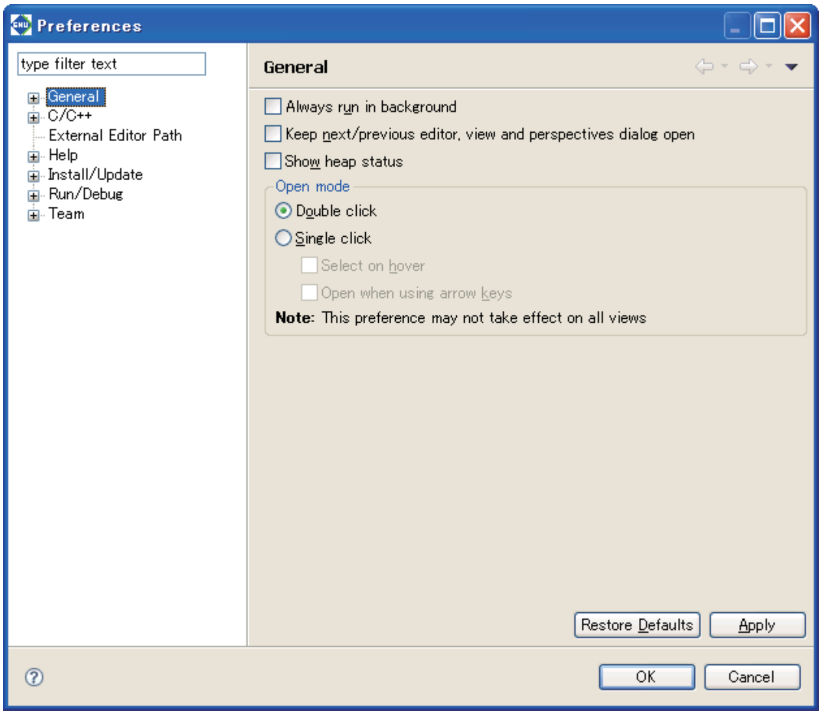
Applies the set content of the current page and closes the dialog box.

[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



☐ Always run in background
☐ Keep next/previous editor, view and perspectives dialog open
☐ Show heap status

Open mode
● Double click
○ Single click
☐ Select on hover
☐ Open when using arrow keys

Note: This preference may not take effect on all views

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.

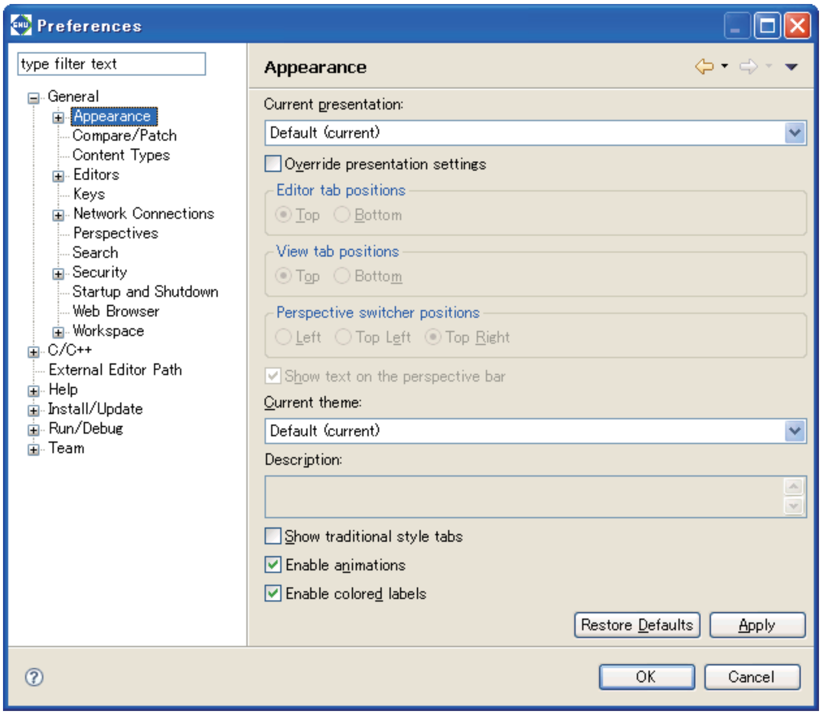
[Select on hover] (default: OFF)

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 > Appearance


Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Keys
Network Connections
Perspectives
Search
Security
Startup and Shutdown
Web Browser
Workspace
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
Appearance
Restore Defaults
Apply
OK
Cancel

Current presentation: Default (current)

☐ Override presentation settings

Editor tab positions: ● Top ○ Bottom

View tab positions: ● Top ○ Bottom

Perspective switcher positions: ○ Left ○ Top Left ● Top Right

☑ Show text on the perspective bar

Current theme: Default (current)

Description:

☐ Show traditional style tabs

☑ Enable animations

☑ Enable colored labels

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.

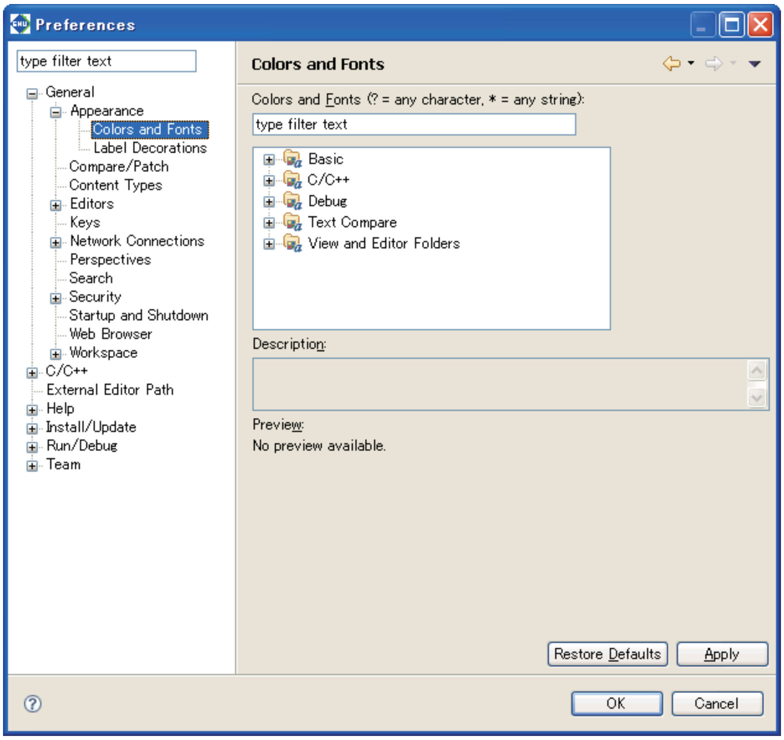
[Show traditional style tabs] (default: OFF)

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 > Appearance > Colors and Fonts


Preferences
type filter text
General
Appearance
Colors and Fonts
Label Decorations
Compare/Patch
Content Types
Editors
Keys
Network Connections
Perspectives
Search
Security
Startup and Shutdown
Web Browser
Workspace
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
Colors and Fonts
Colors and Fonts (? = any character, * = any string):
type filter text
Basic
C/C++
Debug
Text Compare
View and Editor Folders
Description:
Preview:
No preview available.
Restore Defaults
Apply
OK
Cancel

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.

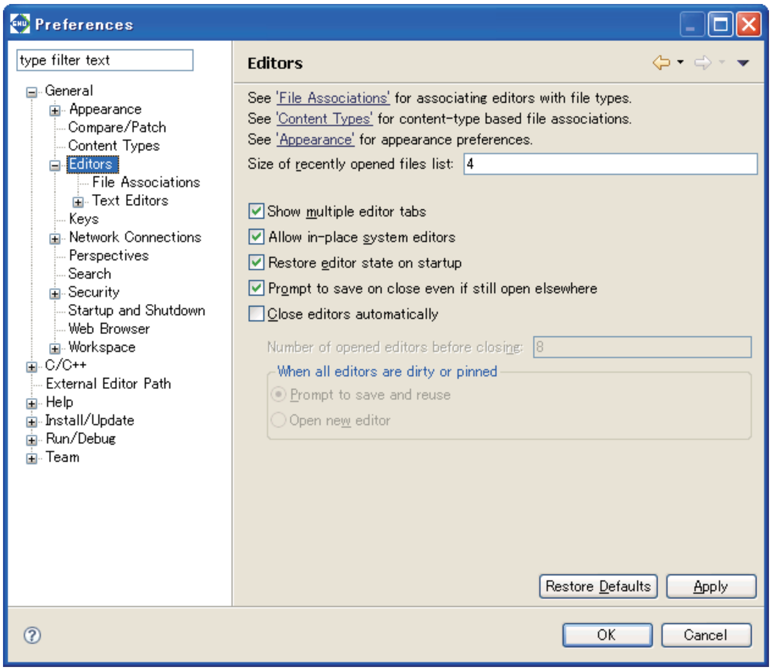
Color select button

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 > Editors


Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
File Associations
Text Editors
Keys
Network Connections
Perspectives
Search
Security
Startup and Shutdown
Web Browser
Workspace
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
Editors
Restore Defaults
Apply
OK
Cancel

See 'File Associations' for associating editors with file types.
See 'Content Types' for content-type based file associations.
See 'Appearance' for appearance preferences.
Size of recently opened files list: 4

☑ Show multiple editor tabs
☑ Allow in-place system editors
☑ Restore editor state on startup
☑ Prompt to save on close even if still open elsewhere
☐ Close editors automatically

Number of opened editors before closing: 8

When all editors are dirty or pinned
● Prompt to save and reuse
○ Open new editor

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.

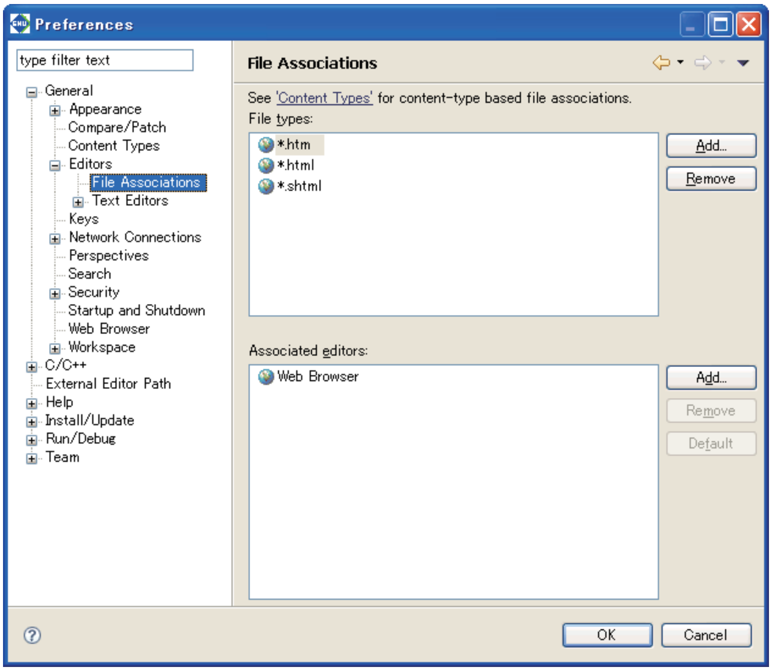
[Prompt to save and reuse] (default: ON)

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 > Editors > File Associations


Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
File Associations
Text Editors
Keys
Network Connections
Perspectives
Search
Security
Startup and Shutdown
Web Browser
Workspace
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
File Associations
See 'Content Types' for content-type based file associations.
File types:
*.htm
*.html
*.shtml
Add...
Remove
Associated editors:
Web Browser
Add...
Remove
Default
OK
Cancel

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.

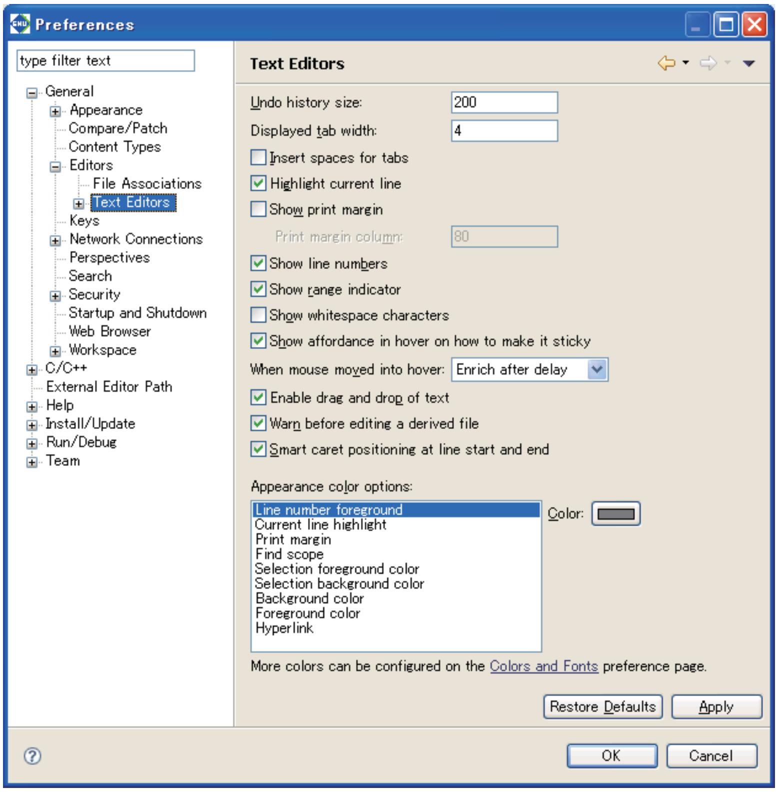
[Remove]

Removes a selected file type or editor from the list box.

[Default]

Sets the editor selected in [Associated editors:] to the default editor.

General > Editors > Text Editors


Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
File Associations
Text Editors
Keys
Network Connections
Perspectives
Search
Security
Startup and Shutdown
Web Browser
Workspace
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
Text Editors
Restore Defaults
Apply
OK
Cancel

Undo history size: 200

Displayed tab width: 4

☐ Insert spaces for tabs

☑ Highlight current line

☐ Show print margin

Print margin column: 80

☑ Show line numbers

☑ Show range indicator

☐ Show whitespace characters

☑ Show affordance in hover on how to make it sticky

When mouse moved into hover: Enrich after delay

☑ Enable drag and drop of text

☑ Warn before editing a derived file

☑ Smart caret positioning at line start and end

Appearance color options:

- Line number foreground
- Current line highlight
- Print margin
- Find scope
- Selection foreground color
- Selection background color
- Background color
- Foreground color
- Hyperlink

Color:

More colors can be configured on the Colors and Fonts preference page.

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: ON)

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 decelerated 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:]

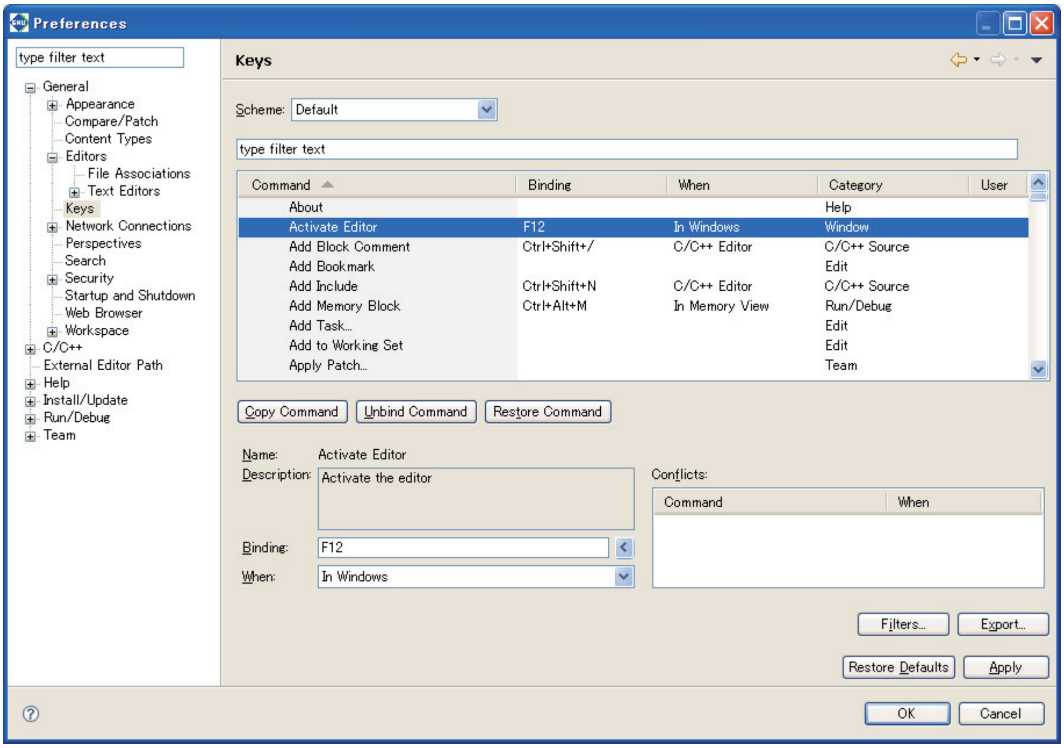
Set the following display colors. Select the item whose display color you want to change from the list, then select a color from the dialog box displayed when you click the [Color:] button.

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.


Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
File Associations
Text Editors
Keys
Network Connections
Perspectives
Search
Security
Startup and Shutdown
Web Browser
Workspace
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
Keys
type filter text
Copy Command
Unbind Command
Restore Command
Conflicts:
Command
When
Filters...
Export...
Restore Defaults
Apply
OK
Cancel

Scheme: Default

Command	Binding	When	Category	User
About			Help	
Activate Editor	F12	In Windows	Window	
Add Block Comment	Ctrl+Shift+/	C/C++ Editor	C/C++ Source	
Add Bookmark			Edit	
Add Include	Ctrl+Shift+N	C/C++ Editor	C/C++ Source	
Add Memory Block	Ctrl+Alt+M	In Memory View	Run/Debug	
Add Task...			Edit	
Add to Working Set			Edit	
Apply Patch...			Team	

Name: Activate Editor

Description: Activate the editor

Binding: F12

When: In Windows

Shows the list of shortcut keys.

[Binding]

A key sequence can be assigned by pressing a key combination.

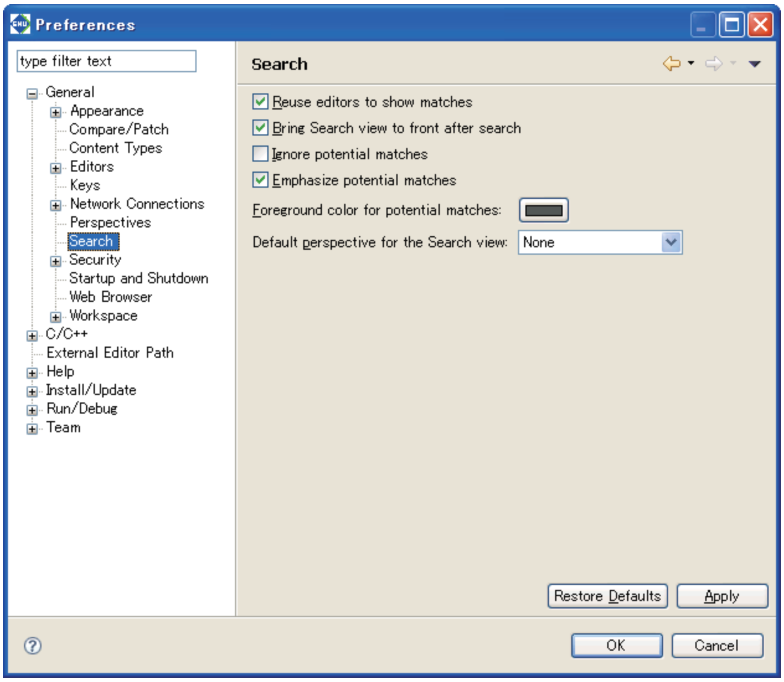
[When:]

Selects a location/state in which the command is enabled when assigning the key sequence set with [Binding] to the command.

[Export...]

Saves the contents of the list to a CSV format file.

General > Search


Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Keys
Network Connections
Perspectives
Search
Security
Startup and Shutdown
Web Browser
Workspace
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
Search
Restore Defaults
Apply
OK
Cancel

☑ Reuse editors to show matches

☑ Bring Search view to front after search

☐ Ignore potential matches

☑ Emphasize potential matches

Foreground color for potential matches: ▬

Default perspective for the Search view: None

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.

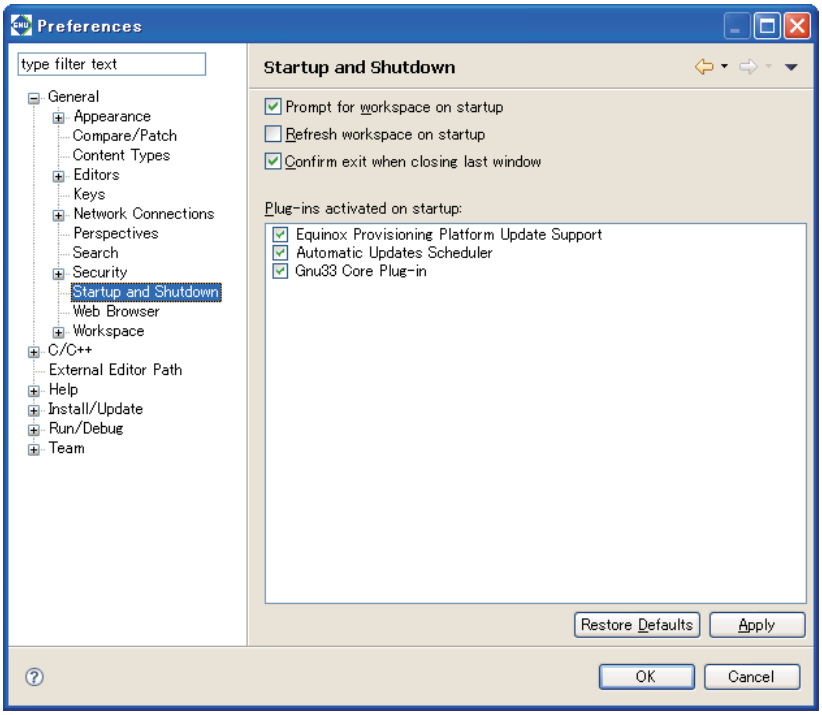
[Bring Search view to front after search] (default: ON)

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.

General > Startup and Shutdown


Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Keys
Network Connections
Perspectives
Search
Security
Startup and Shutdown
Web Browser
Workspace
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
Restore Defaults
Apply
OK
Cancel

Startup and Shutdown

☑ Prompt for workspace on startup
☐ Refresh workspace on startup
☑ Confirm exit when closing last window

Plug-ins activated on startup:

☑ Equinox Provisioning Platform Update Support
☑ Automatic Updates Scheduler
☑ Gnu33 Core Plug-in

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.

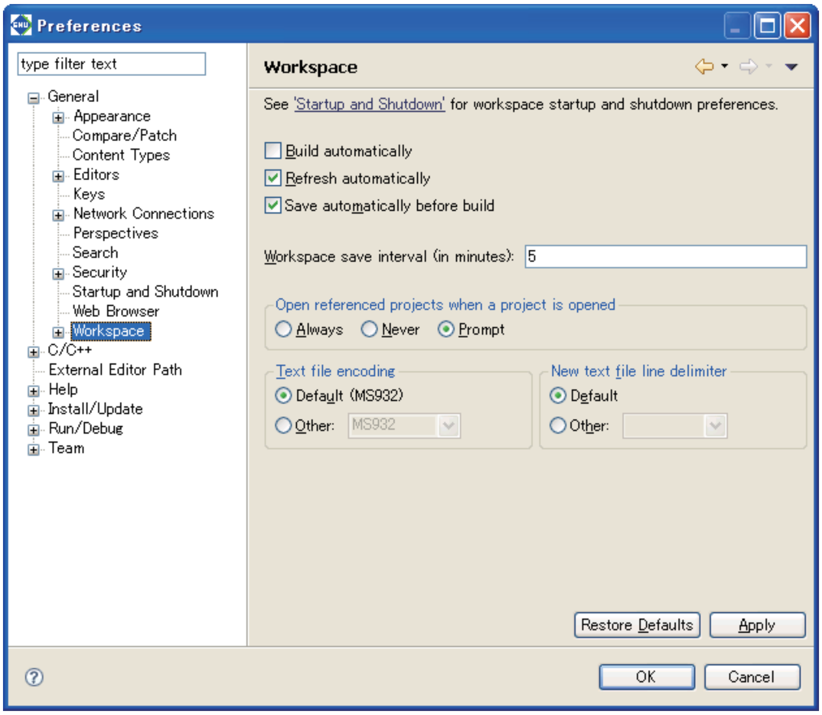
[Confirm exit when closing last window] (default: ON)

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 > Workspace


Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Keys
Network Connections
Perspectives
Search
Security
Startup and Shutdown
Web Browser
Workspace
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
Restore Defaults
Apply
OK
Cancel

Workspace

See 'Startup and Shutdown' for workspace startup and shutdown preferences.

☐ Build automatically
☑ Refresh automatically
☑ Save automatically before build

Workspace save interval (in minutes): 5

Open referenced projects when a project is opened
○ Always ○ Never ● Prompt

Text file encoding
● Default (MS932)
○ Other: MS932

New text file line delimiter
● Default
○ Other:

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.

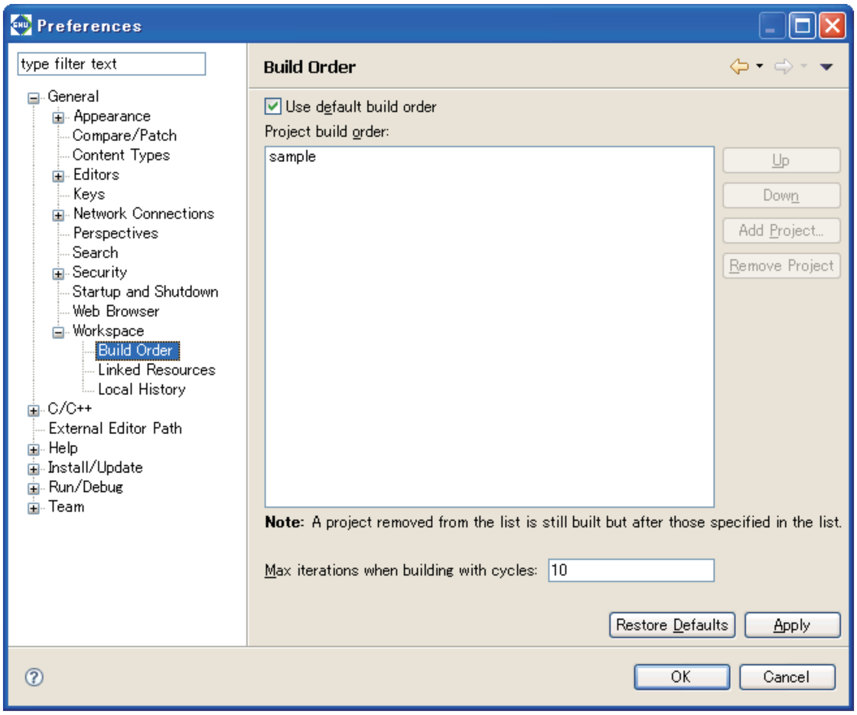
[Default]

The standard Windows line delimiter is used. (default)

[Other:]

Select another line delimiter.

General > Workspace > Build Order


Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Keys
Network Connections
Perspectives
Search
Security
Startup and Shutdown
Web Browser
Workspace
Build Order
Linked Resources
Local History
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
Build Order
Up
Down
Add Project...
Remove Project
Restore Defaults
Apply
OK
Cancel

☑ Use default build order

Project build order:

sample

Note: A project removed from the list is still built but after those specified in the list.

Max iterations when building with cycles: 10

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.

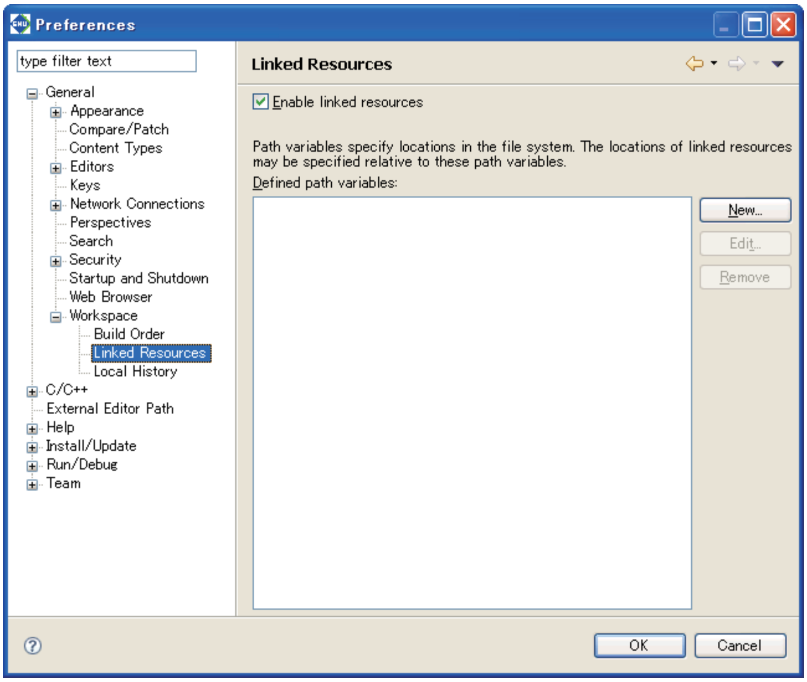
[Add Project...]

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>Workspace>Linked Resources



Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Keys
Network Connections
Perspectives
Search
Security
Startup and Shutdown
Web Browser
Workspace
Build Order
Linked Resources
Local History
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
Linked Resources
New...
Edit...
Remove
OK
Cancel

☑ Enable linked resources

Path variables specify locations in the file system. The locations of linked resources may be specified relative to these path variables.

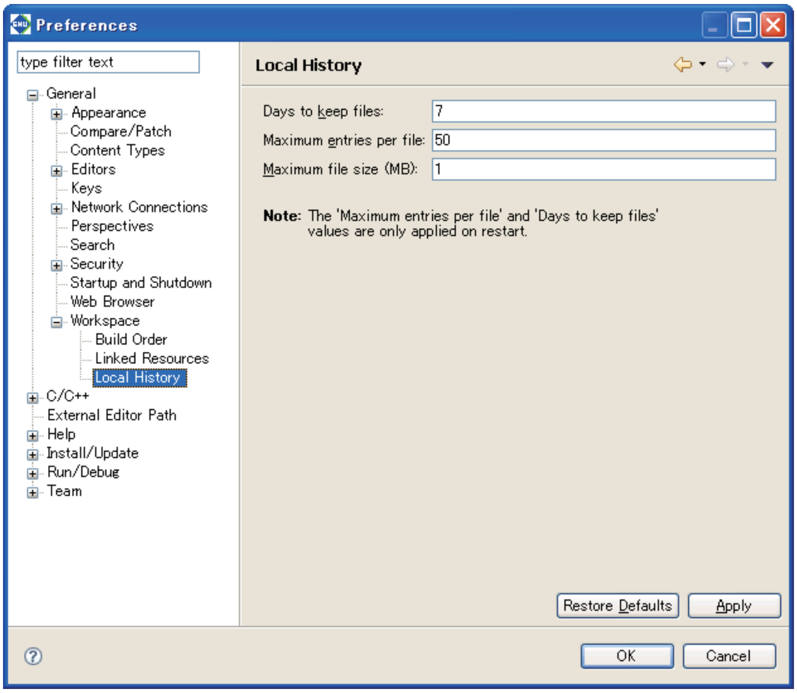
Defined path variables:

Switch the ON/OFF of the link resource.
For detailed information, refer to “Linking to a file located outside the project folder” and “Linking to a folder located outside the project folder” in Section 5.4.8, “Resource Manipulation in a Project.”

[Enable linked resources] (default: ON)

Disables use of the resource linking function when set to OFF.
Do not change this setting in normal use.

General > Workspace > Local History


Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Keys
Network Connections
Perspectives
Search
Security
Startup and Shutdown
Web Browser
Workspace
Build Order
Linked Resources
Local History
C/C++
External Editor Path
Help
Install/Update
Run/Debug
Team
Local History
Restore Defaults
Apply
OK
Cancel

Days to keep files: 7

Maximum entries per file: 50

Maximum file size (MB): 1

Note: The 'Maximum entries per file' and 'Days to keep files' values are only applied on restart.

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)

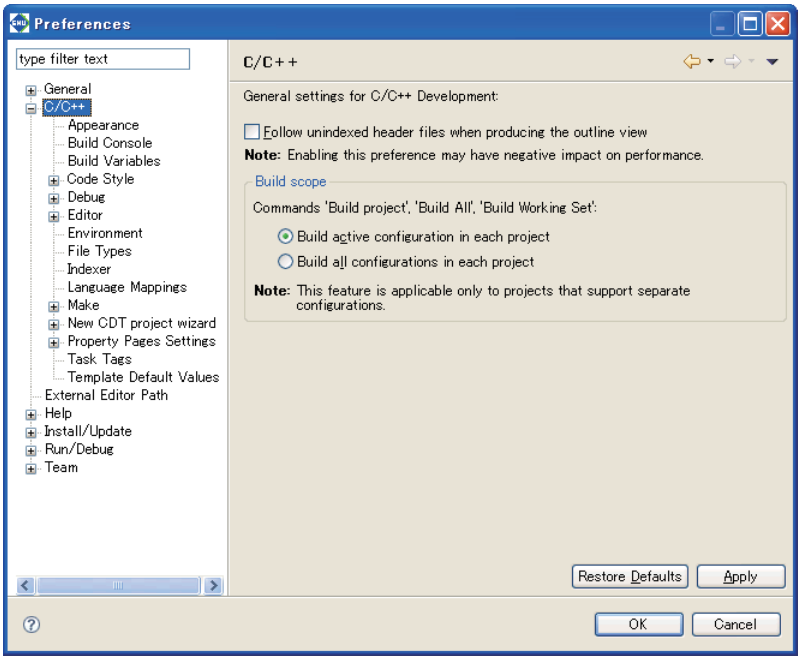
Set the maximum file size for the file containing revision history entries.

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++



Preferences
type filter text
General
C/C++
Appearance
Build Console
Build Variables
Code Style
Debug
Editor
Environment
File Types
Indexer
Language Mappings
Make
New CDT project wizard
Property Pages Settings
Task Tags
Template Default Values
External Editor Path
Help
Install/Update
Run/Debug
Team
C/C++
Restore Defaults
Apply
OK
Cancel

General settings for C/C++ Development:

☐ Follow unindexed header files when producing the outline view

Note: Enabling this preference may have negative impact on performance.

Build scope

Commands 'Build project', 'Build All', 'Build Working Set':

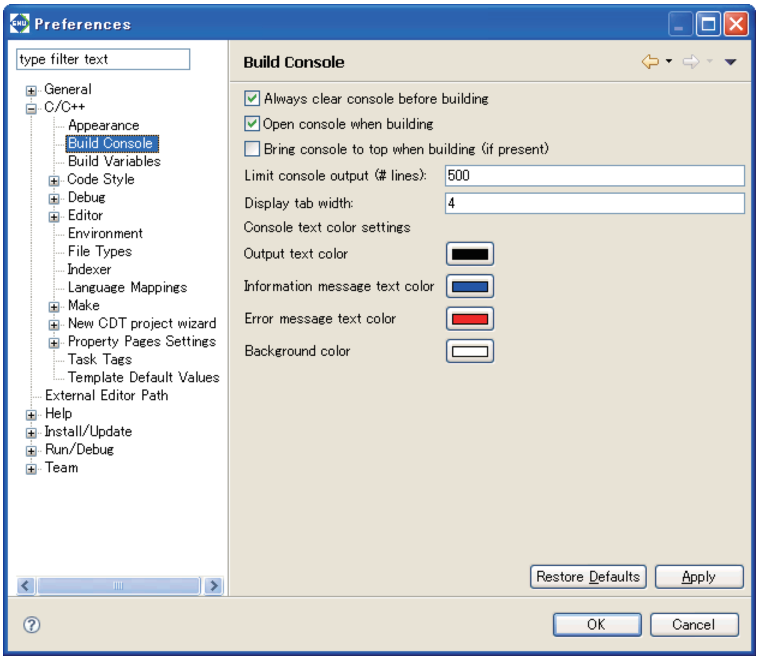
● Build active configuration in each project

○ Build all configurations in each project

Note: This feature is applicable only to projects that support separate configurations.

Disregard this screen.

C/C++ > Build Console


Preferences
type filter text
General
C/C++
Appearance
Build Console
Build Variables
Code Style
Debug
Editor
Environment
File Types
Indexer
Language Mappings
Make
New CDT project wizard
Property Pages Settings
Task Tags
Template Default Values
External Editor Path
Help
Install/Update
Run/Debug
Team
Build Console
Restore Defaults
Apply
OK
Cancel

☑ Always clear console before building
☑ Open console when building
☐ Bring console to top when building (if present)
Limit console output (# lines): 500
Display tab width: 4
Console text color settings
Output text color ⬛
Information message text color 🟦
Error message text color 🟥
Background color ⬜

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]

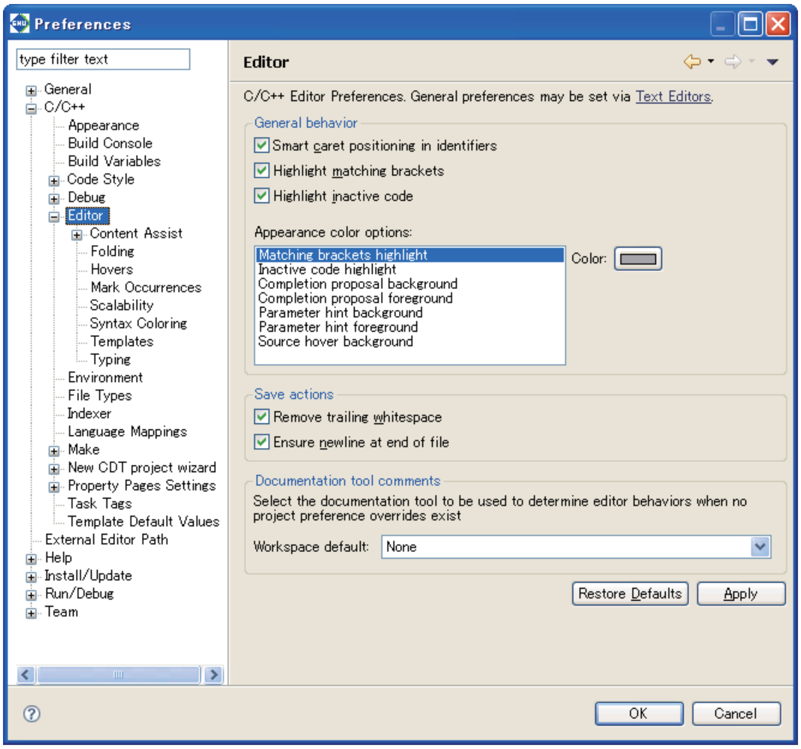
This is the display color for status messages output by tools.

[Error message text color]

This is the display color for error messages output by tools.

C/C++ > Editor

Makes settings for the C editor bundled with the IDE.



Preferences
type filter text
General
C/C++
Appearance
Build Console
Build Variables
Code Style
Debug
Editor
Content Assist
Folding
Hovers
Mark Occurrences
Scalability
Syntax Coloring
Templates
Typing
Environment
File Types
Indexer
Language Mappings
Make
New CDT project wizard
Property Pages Settings
Task Tags
Template Default Values
External Editor Path
Help
Install/Update
Run/Debug
Team
Editor
Restore Defaults
Apply
OK
Cancel

C/C++ Editor Preferences. General preferences may be set via Text Editors.

General behavior

☑ Smart caret positioning in identifiers

☑ Highlight matching brackets

☑ Highlight inactive code

Appearance color options:

- Matching brackets highlight
- Inactive code highlight
- Completion proposal background
- Completion proposal foreground
- Parameter hint background
- Parameter hint foreground
- Source hover background

Color:

Save actions

☑ Remove trailing whitespace

☑ Ensure newline at end of file

Documentation tool comments

Select the documentation tool to be used to determine editor behaviors when no project preference overrides exist

Workspace default: None

[General behavior]

[Smart caret positioning in identifiers] (default: ON)

Select this checkbox to display a border between words in identifiers.

[Highlight matching brackets] (default: ON)

If the cursor is placed on a line with parentheses ( ) (or braces { }) with this checkbox selected, the corresponding parentheses will be surrounded with a border.

[Highlight inactive code] (default: ON)

If the cursor is placed on a line with inactive code with this checkbox selected, the inactive code is highlighted.

[Appearance color options:]

Sets the color for characters in comments or keywords. Select the statement type from the list and select the desired color using the [Color] button.

[Save Actions]

[Remove trailing whitespace]

Deletes blank space from the end of line when saving data to a file.

[Ensure newline at end of file]

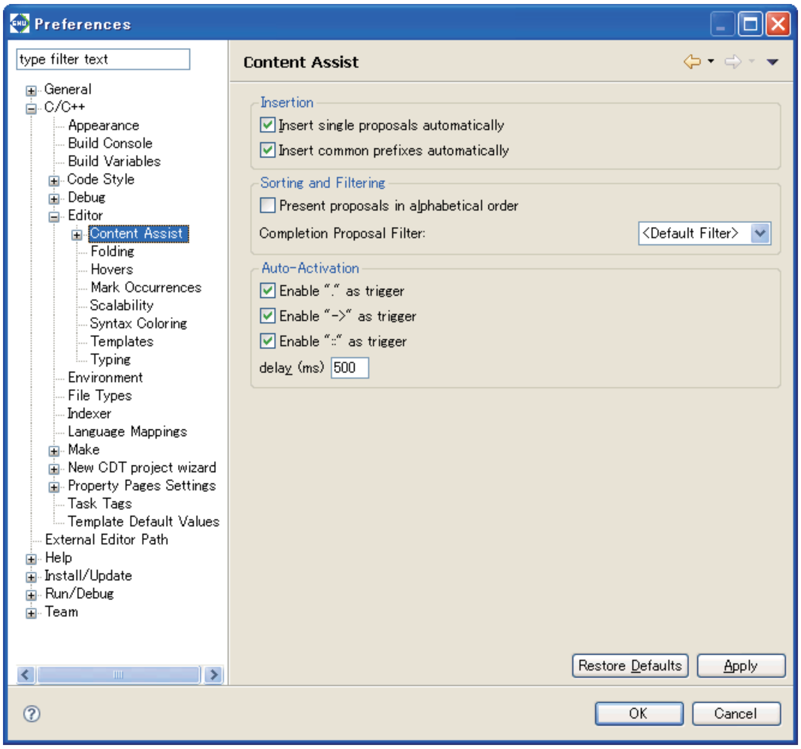
Enters a line break at the end of the last line when saving data to a file.

[Documentation tool comments]

Select the type of documentation tool that determines the editor display and operation. Set editor functions such as content assist, color coding, and comment generation based on the tool selected here.

The settings selected here will be applied to files unrelated to the project or files without project level settings.

C/C++ > Editor > Content Assist


Preferences
type filter text
General
C/C++
Appearance
Build Console
Build Variables
Code Style
Debug
Editor
Content Assist
Folding
Hovers
Mark Occurrences
Scalability
Syntax Coloring
Templates
Typing
Environment
File Types
Indexer
Language Mappings
Make
New CDT project wizard
Property Pages Settings
Task Tags
Template Default Values
External Editor Path
Help
Install/Update
Run/Debug
Team
Restore Defaults
Apply
OK
Cancel

Content Assist

Insertion

☑ Insert single proposals automatically

☑ Insert common prefixes automatically

Sorting and Filtering

☐ Present proposals in alphabetical order

Completion Proposal Filter: <Default Filter>

Auto-Activation

☑ Enable "." as trigger

☑ Enable "->" as trigger

☑ Enable "::" as trigger

delay (ms) 500

Makes settings for content assist.
Disregard in normal use.

[Insertion]

Makes settings for the display and insertion of candidates.

[Insert single proposals automatically] (default: ON)

Select this checkbox to enter the candidate automatically when there is only one candidate.

[Insert common prefixes automatically] (default: ON)

Select this checkbox to enter common prefixes automatically.

[Sorting and Filtering]

Makes settings for sorting and filtering.

[Present proposals in alphabetical order] (default: OFF)

Select this checkbox to list candidates in the content assist list in alphabetical order. Deselect the checkbox to list candidates in order of suitability determined by factors such as relative position, scope, and prefix.

[Auto activation:]

Automatically starts up content assist when a symbol (“.,” “->,” “::”) is entered.
To disable this function, deselect the checkbox.

[Enable "." as trigger] (default: ON)

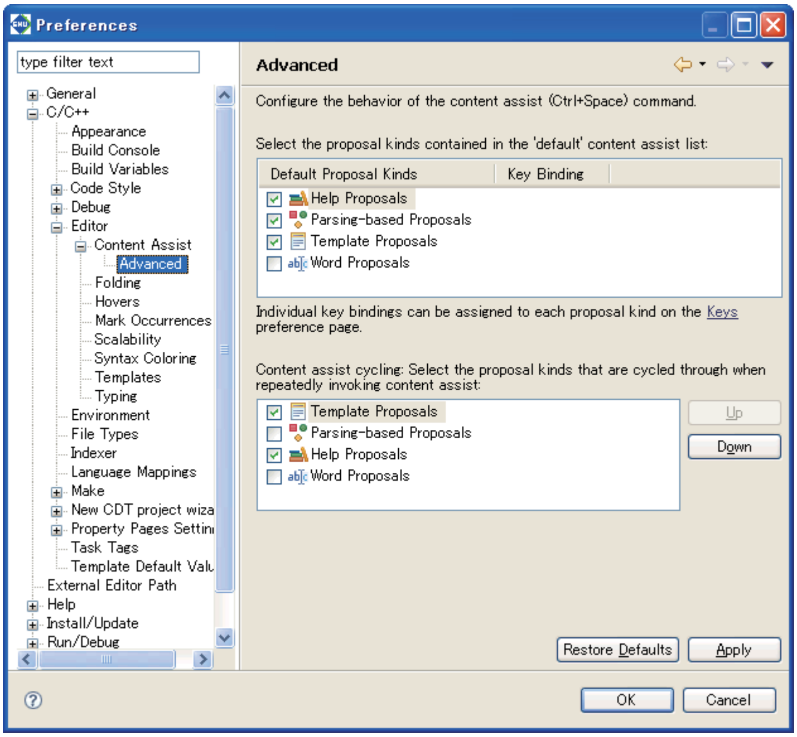
[Enable "->" as trigger] (default: ON)

[Enable "::" as trigger] (default: ON)

[delay (ms)] (default: 500 ms)

Specify the length of time (in milliseconds) from the entry of a symbol described above to automatic display of the content assist.

C/C++ > Editor > Content assist > Advanced



Preferences
type filter text
General
C/C++
Appearance
Build Console
Build Variables
Code Style
Debug
Editor
Content Assist
Advanced
Folding
Hovers
Mark Occurrences
Scalability
Syntax Coloring
Templates
Typing
Environment
File Types
Indexer
Language Mappings
Make
New CDT project wiza
Property Pages Settin
Task Tags
Template Default Valu
External Editor Path
Help
Install/Update
Run/Debug
Advanced
Default Proposal Kinds
Key Binding
Up
Down
Restore Defaults
Apply
OK
Cancel

Configure the behavior of the content assist (Ctrl+Space) command.

Select the proposal kinds contained in the 'default' content assist list:

- ☑ Help Proposals
- ☑ Parsing-based Proposals
- ☑ Template Proposals
- ☐ Word Proposals

Individual key bindings can be assigned to each proposal kind on the Keys preference page.

Content assist cycling: Select the proposal kinds that are cycled through when repeatedly invoking content assist:

- ☑ Template Proposals
- ☐ Parsing-based Proposals
- ☑ Help Proposals
- ☐ Word Proposals

Makes advanced settings for the content assist.

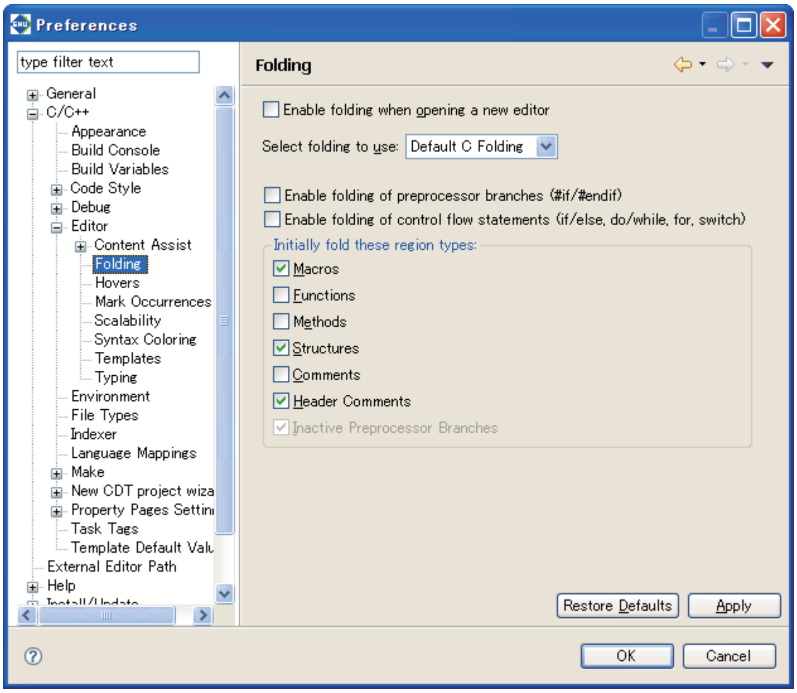
[Select the proposal kinds contained in the 'default' content assist list:]

Select the functions to be included in the content assist as standard functions.

[Content assist cycling: Select the proposal kinds that are cycled through when repeatedly invoking content assist:]

Select the functions to be added when the content assist is called repeatedly. Select an item and click the [Up] or [Down] button to set the priority order of that item.

C/C++ > Editor > Folding



Preferences
type filter text
General
C/C++
Appearance
Build Console
Build Variables
Code Style
Debug
Editor
Content Assist
Folding
Hovers
Mark Occurrences
Scalability
Syntax Coloring
Templates
Typing
Environment
File Types
Indexer
Language Mappings
Make
New CDT project wiza
Property Pages Settin
Task Tags
Template Default Valu
External Editor Path
Help
Install/Update
Restore Defaults
Apply
OK
Cancel

Folding

☐ Enable folding when opening a new editor

Select folding to use: Default C Folding

☐ Enable folding of preprocessor branches (#if/#endif)

☐ Enable folding of control flow statements (if/else, do/while, for, switch)

Initially fold these region types:

☑ Macros

☐ Functions

☐ Methods

☑ Structures

☐ Comments

☑ Header Comments

☑ Inactive Preprocessor Branches

Makes settings for the folding function.

[Enable folding when opening a new editor] (default: OFF)

Select this checkbox to apply the folding function to the newly opened editor.

[Select folding to use:] (default: OFF)

Select the type of folding function.

[Enable folding of preprocessor branches (#if/#endif)] (default: OFF)

Select this checkbox to fold preprocessor branches.

[Enable folding of control flow statements (if/else, do/while, for, switch)] (default: OFF)

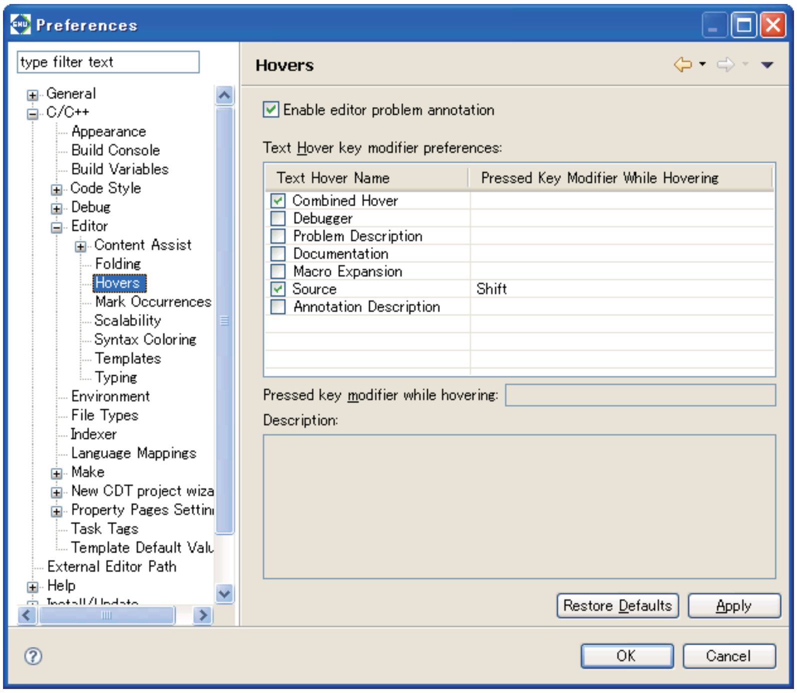
Select this checkbox to fold flow control declaration statements.

[Initially fold these region types:]

Select initially fold types when opening an editor.

[Macros] (default: ON)	Macros
[Functions] (default: OFF)	Functions
[Methods] (default: OFF)	Methods
[Structures] (default: ON)	Structures
[Comments] (default: OFF)	Comments
[Header Comments] (default: ON)	Headers
[Inactive Preprocessor Branches] (default: ON)	Inactive preprocessor branches

C/C++ > Editor > Hovers


Preferences
type filter text
General
C/C++
Appearance
Build Console
Build Variables
Code Style
Debug
Editor
Content Assist
Folding
Hovers
Mark Occurrences
Scalability
Syntax Coloring
Templates
Typing
Environment
File Types
Indexer
Language Mappings
Make
New CDT project wiza
Property Pages Settin
Task Tags
Template Default Valu
External Editor Path
Help
Install/Update
Hovers
Restore Defaults
Apply
OK
Cancel

☑ Enable editor problem annotation

Text Hover key modifier preferences:

Text Hover Name	Pressed Key Modifier While Hovering
☑ Combined Hover	
☐ Debugger	
☐ Problem Description	
☐ Documentation	
☐ Macro Expansion	
☑ Source	Shift
☐ Annotation Description	

Pressed key modifier while hovering:

Description:

Makes settings for a dialog balloon.

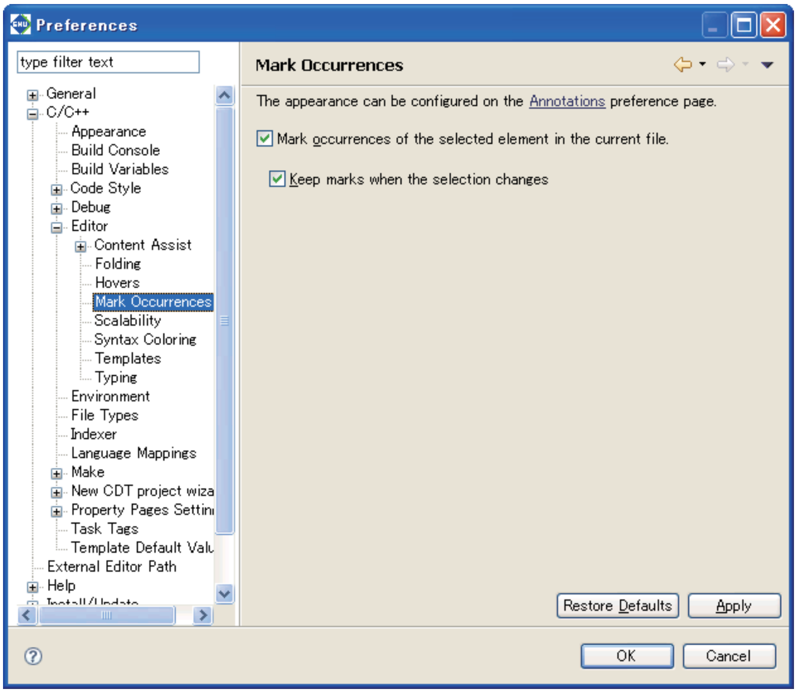
[Enable editor problem annotation] (default: ON)

Select this checkbox to highlight discovered problems.

[Text Hover key modifier preferences:]

Select a hot key to activate text hover. For example, positioning the mouse cursor while pressing the [Ctrl] key links the target to the original declaration.

C/C++ > Editor > Mark Occurrences


Preferences
type filter text
General
C/C++
Appearance
Build Console
Build Variables
Code Style
Debug
Editor
Content Assist
Folding
Hovers
Mark Occurrences
Scalability
Syntax Coloring
Templates
Typing
Environment
File Types
Indexer
Language Mappings
Make
New CDT project wiza
Property Pages Settin
Task Tags
Template Default Valu
External Editor Path
Help
Install/Update
Restore Defaults
Apply
OK
Cancel

Mark Occurrences

The appearance can be configured on the Annotations preference page.

☑ Mark occurrences of the selected element in the current file.

☑ Keep marks when the selection changes

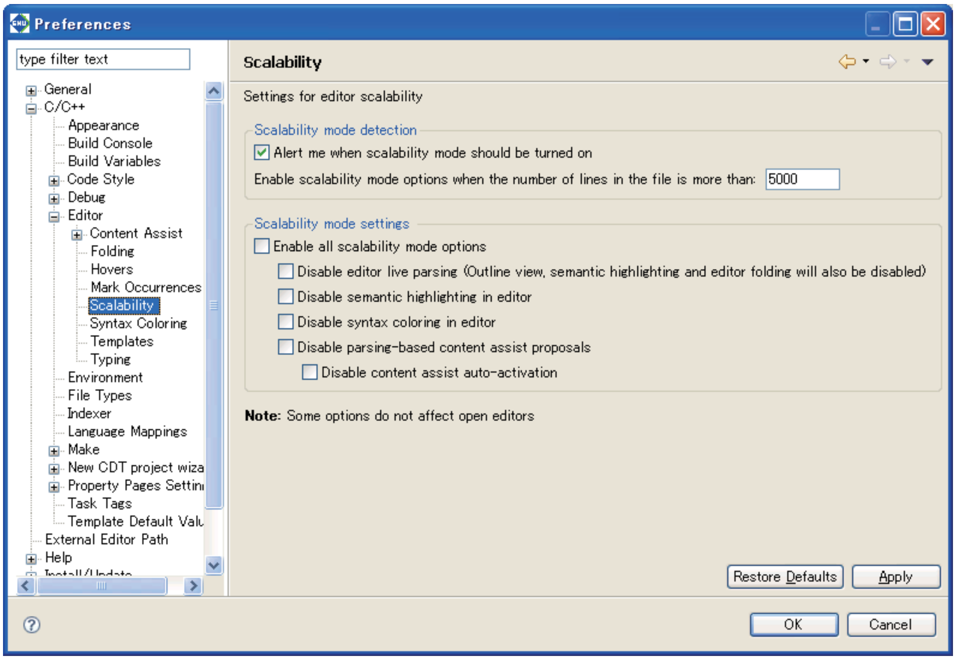
[Mark occurrences of the selected element in the current file:] (default: ON)

Select this checkbox to display variables, functions, methods, types, macros, or other component marks.

[Keep marks when the selection changes] (default: ON)

Select this checkbox to retain marks even if the selection is changed.

C/C++ > Editor > Scalability



Scalability

Settings for editor scalability

Scalability mode detection

☑ Alert me when scalability mode should be turned on

Enable scalability mode options when the number of lines in the file is more than: 5000

Scalability mode settings

☐ Enable all scalability mode options

☐ Disable editor live parsing (Outline view, semantic highlighting and editor folding will also be disabled)

☐ Disable semantic highlighting in editor

☐ Disable syntax coloring in editor

☐ Disable parsing-based content assist proposals

☐ Disable content assist auto-activation

Note: Some options do not affect open editors

Preferences
type filter text
General
C/C++
Appearance
Build Console
Build Variables
Code Style
Debug
Editor
Content Assist
Folding
Hovers
Mark Occurrences
Scalability
Syntax Coloring
Templates
Typing
Environment
File Types
Indexer
Language Mappings
Make
New CDT project wiza
Property Pages Settin
Task Tags
Template Default Valu
External Editor Path
Help
Install/Update
Restore Defaults
Apply
OK
Cancel

Adjusts the performance of an editor.

This function restricts editor display functions to prevent degradation of editor performance including highlight function when the editor opens a file containing many lines.

This can generally be disregarded in normal use.

[Scalability mode detection]

[Alert me when scalability mode should be turned on]

By default, a dialog box opens to issue a warning about editor performance when a file source with 5,000 lines or more is opened with an editor.

[Scalability mode settings]

[Enable all scalability mode options]

Enables option for restricting editor functions when a large file is opened.

[Disable editor live parsing]

Disables the editor live syntax analysis function (also disables outline, highlight, and fold display functions).

[Disable semantic highlighting in editor]

Disables the editor highlighting feature.

[Disable syntax coloring in editor]

Disables the editor syntax coloring function.

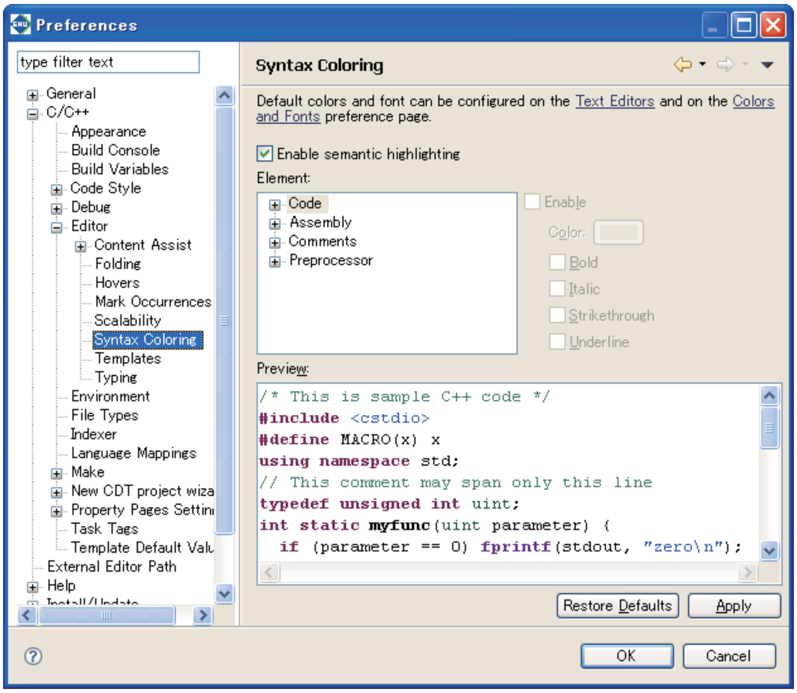
[Disable parsing-based content assist proposal]

Disables the syntax-analysis-based content assist function.

[Disable content assist auto-activation]

Disables the content assist automatic launch function.

C/C++ > Editor > Syntax Coloring


Preferences
type filter text
General
C/C++
Appearance
Build Console
Build Variables
Code Style
Debug
Editor
Content Assist
Folding
Hovers
Mark Occurrences
Scalability
Syntax Coloring
Templates
Typing
Environment
File Types
Indexer
Language Mappings
Make
New CDT project wiza
Property Pages Settin
Task Tags
Template Default Valu
External Editor Path
Help
Install/Update
Syntax Coloring
Element:
Code
Assembly
Comments
Preprocessor
Enable
Color:
Bold
Italic
Strikethrough
Underline
Preview:
Restore Defaults
Apply
OK
Cancel

Default colors and font can be configured on the Text Editors and on the Colors and Fonts preference page.

☑ Enable semantic highlighting

```
/* This is sample C++ code */
#include <cstdio>
#define MACRO(x) x
using namespace std;
// This comment may span only this line
typedef unsigned int uint;
int static myfunc(uint parameter) {
  if (parameter == 0) fprintf(stdout, "zero\n");
```

Makes a highlight setting for syntax.

[Enable semantic highlighting:] (default: ON)

Select this checkbox to enable color settings.

[Element:]

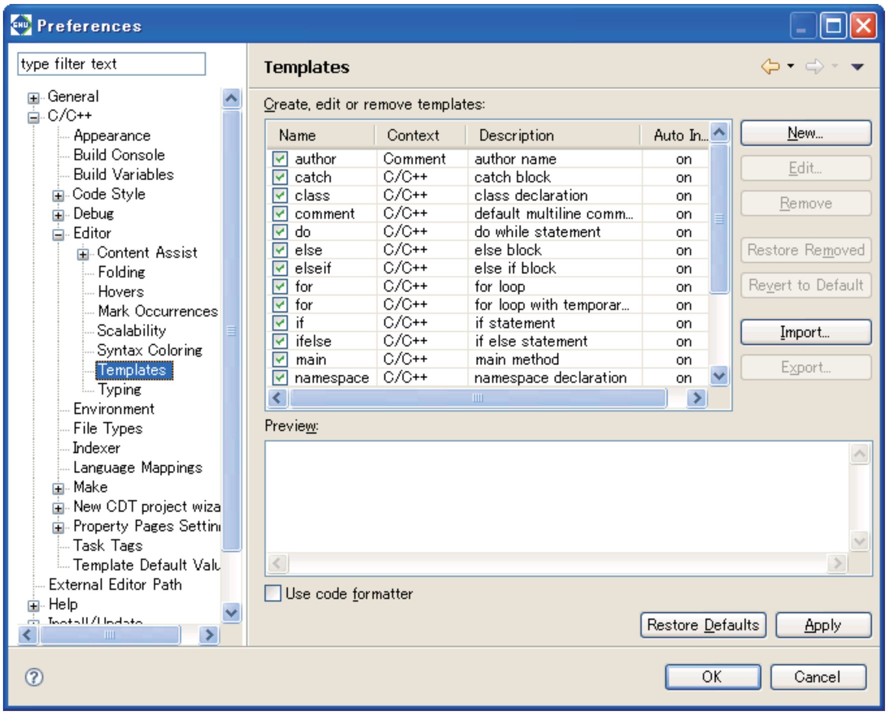
Sets colors for characters in comments or keywords. Select the statement type from the list and select the desired color using the [Color] button. The following checkboxes will modify character style attributes.

[Bold]	Bold
[Italic]	Italic
[Strikethough]	Strikethrough
[Underline]	Underline

[Preview:]

Displays a sample to confirm the settings above.

C/C++ > Editor > Templates


Preferences
type filter text
General
C/C++
Appearance
Build Console
Build Variables
Code Style
Debug
Editor
Content Assist
Folding
Hovers
Mark Occurrences
Scalability
Syntax Coloring
Templates
Typing
Environment
File Types
Indexer
Language Mappings
Make
New CDT project wiza
Property Pages Settin
Task Tags
Template Default Valu
External Editor Path
Help
Install/Update
Templates
Create, edit or remove templates:
New...
Edit...
Remove
Restore Removed
Revert to Default
Import...
Export...
Preview:
Use code formatter
Restore Defaults
Apply
OK
Cancel

Name	Context	Description	Auto In...
☑ author	Comment	author name	on
☑ catch	C/C++	catch block	on
☑ class	C/C++	class declaration	on
☑ comment	C/C++	default multiline comm...	on
☑ do	C/C++	do while statement	on
☑ else	C/C++	else block	on
☑ elseif	C/C++	else if block	on
☑ for	C/C++	for loop	on
☑ for	C/C++	for loop with temporar...	on
☑ if	C/C++	if statement	on
☑ ifelse	C/C++	if else statement	on
☑ main	C/C++	main method	on
☑ namespace	C/C++	namespace declaration	on

☐ Use code formatter

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/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.

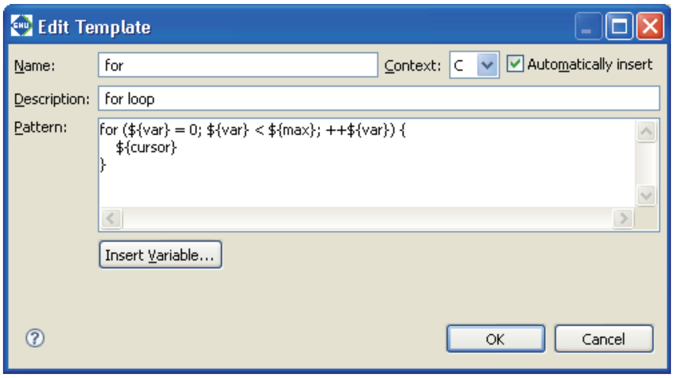
[New...]

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



Name: for

Context: C

☑ Automatically insert

Description: for loop

Pattern:

```
for (${var} = 0; ${var} < ${max}; ++${var}) {
   ${cursor}
}
```

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.

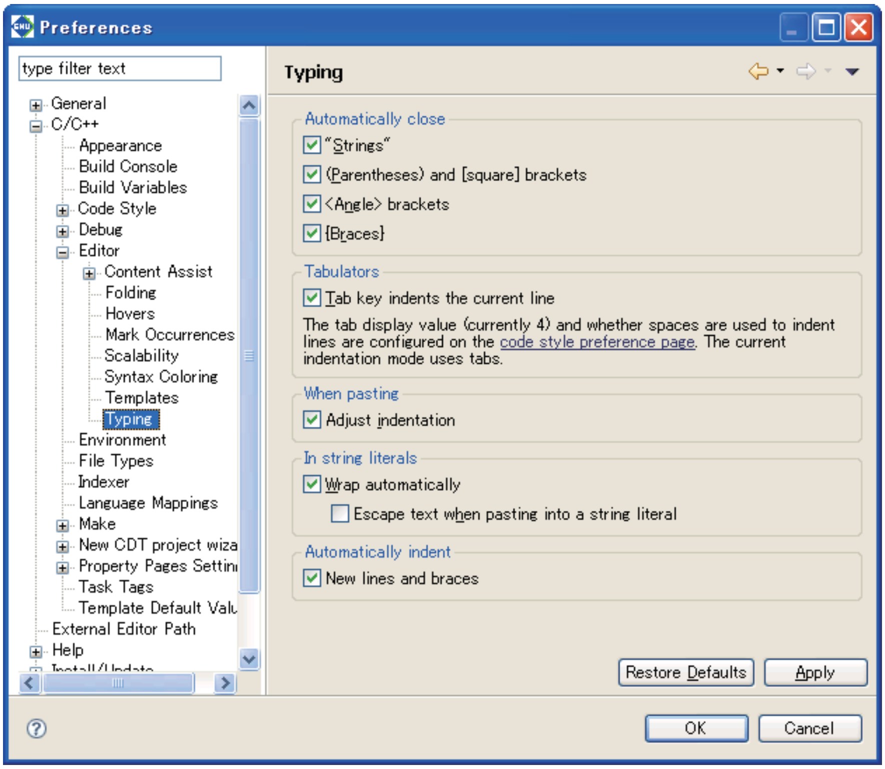
[Import...]

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.

Editor > Typing


Preferences
type filter text
General
C/C++
Appearance
Build Console
Build Variables
Code Style
Debug
Editor
Content Assist
Folding
Hovers
Mark Occurrences
Scalability
Syntax Coloring
Templates
Typing
Environment
File Types
Indexer
Language Mappings
Make
New CDT project wiza
Property Pages Settin
Task Tags
Template Default Valu
External Editor Path
Help
Install/Update
Typing
Restore Defaults
Apply
OK
Cancel

Automatically close

☑ "Strings"

☑ (Parentheses) and [square] brackets

☑ <Angle> brackets

☑ {Braces}

Tabulators

☑ Tab key indents the current line

The tab display value (currently 4) and whether spaces are used to indent lines are configured on the code style preference page. The current indentation mode uses tabs.

When pasting

☑ Adjust indentation

In string literals

☑ Wrap automatically

☐ Escape text when pasting into a string literal

Automatically indent

☑ New lines and braces

Makes settings for typing (entering data).

[Automatically close]

Closes the following items automatically.

["Strings"] (default: ON)

Select this checkbox to close strings automatically.

[(Parenthesis) and [square] brackets] (default: ON)

Select this checkbox to close parentheses ( ) and brackets [ ] automatically.

[<Angle> brackets] (default: ON)

Select this checkbox to close angle brackets < > automatically.

[{Braces}] (default: ON)

Select this checkbox to close braces { } automatically.

[Tabulators]

Sets the operation of the [Tab] key.

[Tab key indents the current line] (default: ON)

Select this checkbox to indent lines on which the [Tab] key is pressed.

[When pasting]

Sets the process to be performed by pasting.

[Adjust indentation] (default: ON)

Select this checkbox to adjust indents in the pasted text to the current indent style.

[In string literals]

Sets processing for strings.

[Wrap automatically] (default: ON)

Select this checkbox to wrap strings automatically if they exceed the maximum number of characters per line.

[Escape text when pasting into a string literal] (default: OFF)

Select this checkbox to enclose special characters in pasted strings with escape characters automatically.

[Automatically indent]

Sets automatic indentation.

[New lines and braces] (default: ON)

Select this checkbox to indent new lines and braces { } automatically.

Other setting items

In normal use, disregard the following items. (These items are not discussed in this manual.)

Environment
File Types
Indexer
Language Mappings
Make
New CDT Project Wizard
Property Page Settings
Task Tags
Template Default Values

5.10 Additional Description on Dialog Boxes

This section discuss 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] *

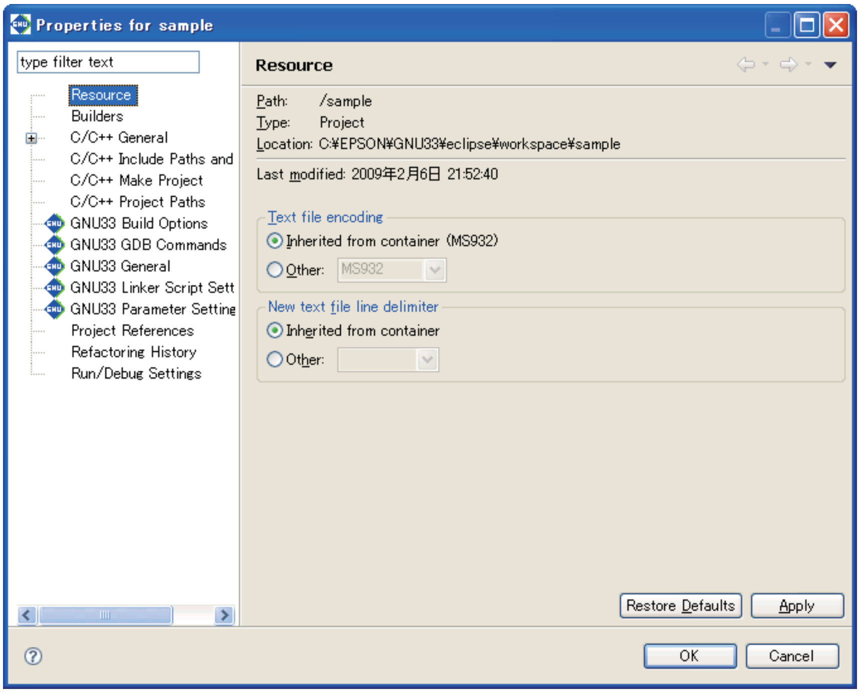
Applies the changes and content set on the current page and closes the dialog box.

[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 [GNU33 Build Options] or [GNU33 General] page have been changed, a dialog box appears for selecting "`clean`" build (see Section 5.7.9) to delete the files created with the previous settings (and rebuild).

Resource


Properties for sample
type filter text
Resource
Builders
C/C++ General
C/C++ Include Paths and
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Commands
GNU33 General
GNU33 Linker Script Sett
GNU33 Parameter Setting
Project References
Refactoring History
Run/Debug Settings
Restore Defaults
Apply
OK
Cancel

Resource

Path: /sample
Type: Project
Location: C:¥EPSON¥GNU33¥eclipse¥workspace¥sample

Last modified: 2009年2月6日 21:52:40

Text file encoding
● Inherited from container (MS932)
○ Other: MS932

New text file line delimiter
● Inherited from container
○ Other:

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.

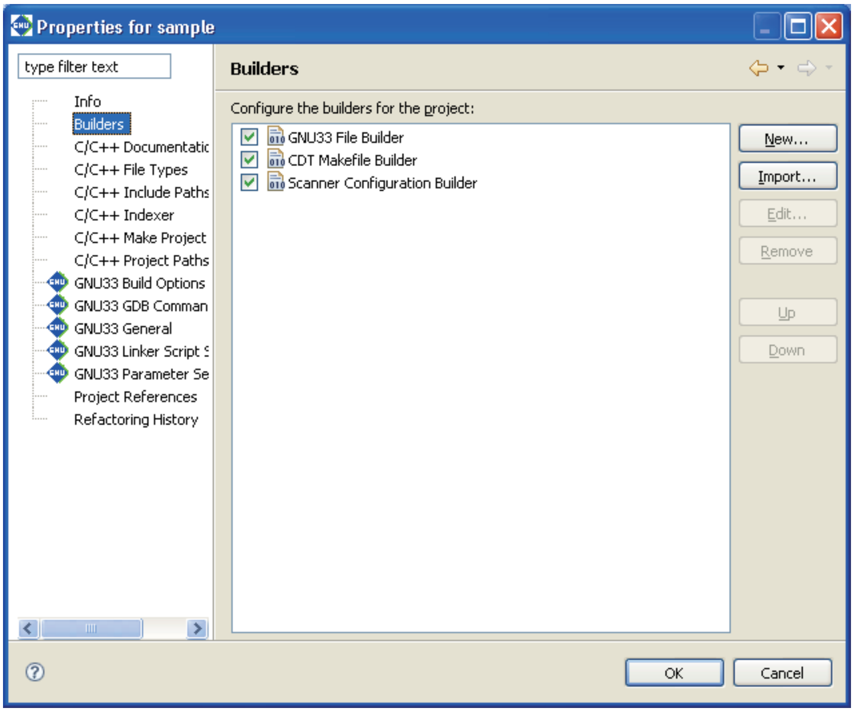
[Inherited from container]

The standard Windows line delimiter is used. (default)

[Other:]

Select another line delimiter.

Builders


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
Builders
New...
Import...
Edit...
Remove
Up
Down
OK
Cancel

Configure the builders for the project:

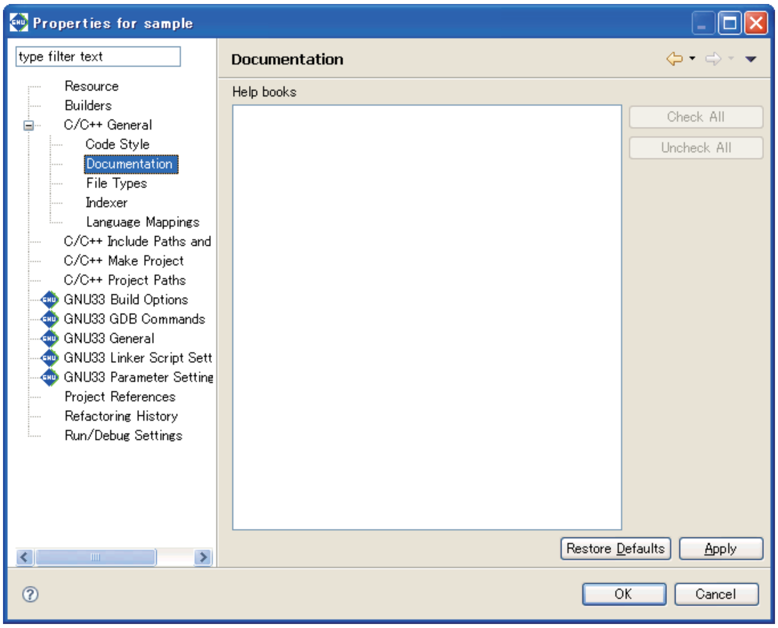
☑ GNU33 File Builder
☑ CDT Makefile Builder
☑ Scanner Configuration Builder

Select the builder for a project.

[GNU33 File Builder]

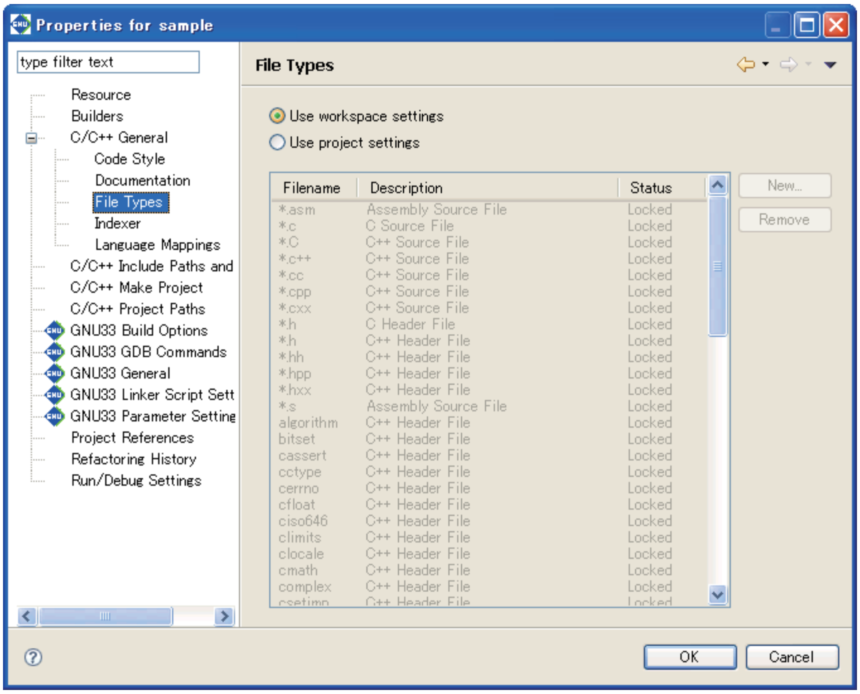
Deselect this check box if you do not wish to automatically generate the makefiles and other files needed for a build when executing a build process. (Refer to Section 5.7.10, "Using an Original Makefile".) Otherwise, avoid making any other changes on this page.

C/C++ General > Documentation


Properties for sample
type filter text
Resource
Builders
C/C++ General
Code Style
Documentation
File Types
Indexer
Language Mappings
C/C++ Include Paths and
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Commands
GNU33 General
GNU33 Linker Script Sett
GNU33 Parameter Setting
Project References
Refactoring History
Run/Debug Settings
Documentation
Help books
Check All
Uncheck All
Restore Defaults
Apply
OK
Cancel

Used to select the help documents for the project. No document is used in the **IDE**.

C/C++ General > File Types


Properties for sample
type filter text
Resource
Builders
C/C++ General
Code Style
Documentation
File Types
Indexer
Language Mappings
C/C++ Include Paths and
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Commands
GNU33 General
GNU33 Linker Script Sett
GNU33 Parameter Setting
Project References
Refactoring History
Run/Debug Settings
File Types
New...
Remove
OK
Cancel

● Use workspace settings
○ Use project settings

Filename	Description	Status
*.asm	Assembly Source File	Locked
*.c	C Source File	Locked
*.C	C++ Source File	Locked
*.c++	C++ Source File	Locked
*.cc	C++ Source File	Locked
*.cpp	C++ Source File	Locked
*.cxx	C++ Source File	Locked
*.h	C Header File	Locked
*.h	C++ Header File	Locked
*.hh	C++ Header File	Locked
*.hpp	C++ Header File	Locked
*.hxx	C++ Header File	Locked
*.s	Assembly Source File	Locked
algorithm	C++ Header File	Locked
bitset	C++ Header File	Locked
cassert	C++ Header File	Locked
cctype	C++ Header File	Locked
cerrno	C++ Header File	Locked
cfloat	C++ Header File	Locked
ciso646	C++ Header File	Locked
climits	C++ Header File	Locked
clocale	C++ Header File	Locked
cmath	C++ Header File	Locked
complex	C++ Header File	Locked
csetjmp	C++ Header File	Locked

Define the name or extension of the file you want to be handled as a C/C++ resource. There is no need to make changes if you use only files created in the **IDE**.

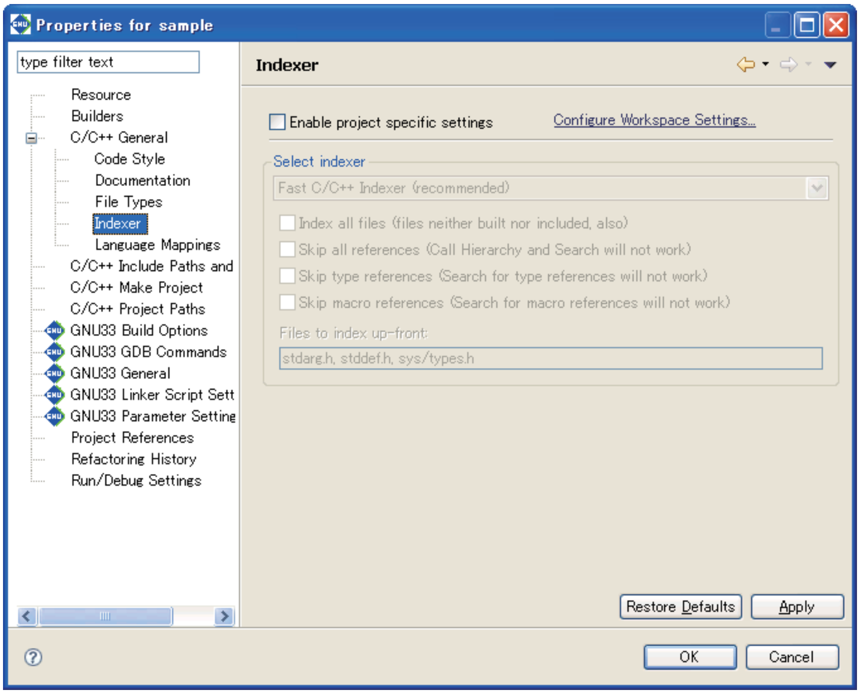
[Use workspace settings]

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++ General > Indexer


Properties for sample
type filter text
Resource
Builders
C/C++ General
Code Style
Documentation
File Types
Indexer
Language Mappings
C/C++ Include Paths and
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Commands
GNU33 General
GNU33 Linker Script Sett
GNU33 Parameter Setting
Project References
Refactoring History
Run/Debug Settings
Indexer
Configure Workspace Settings...
Restore Defaults
Apply
OK
Cancel

☐ Enable project specific settings

Select indexer

Fast C/C++ Indexer (recommended)

☐ Index all files (files neither built nor included, also)

☐ Skip all references (Call Hierarchy and Search will not work)

☐ Skip type references (Search for type references will not work)

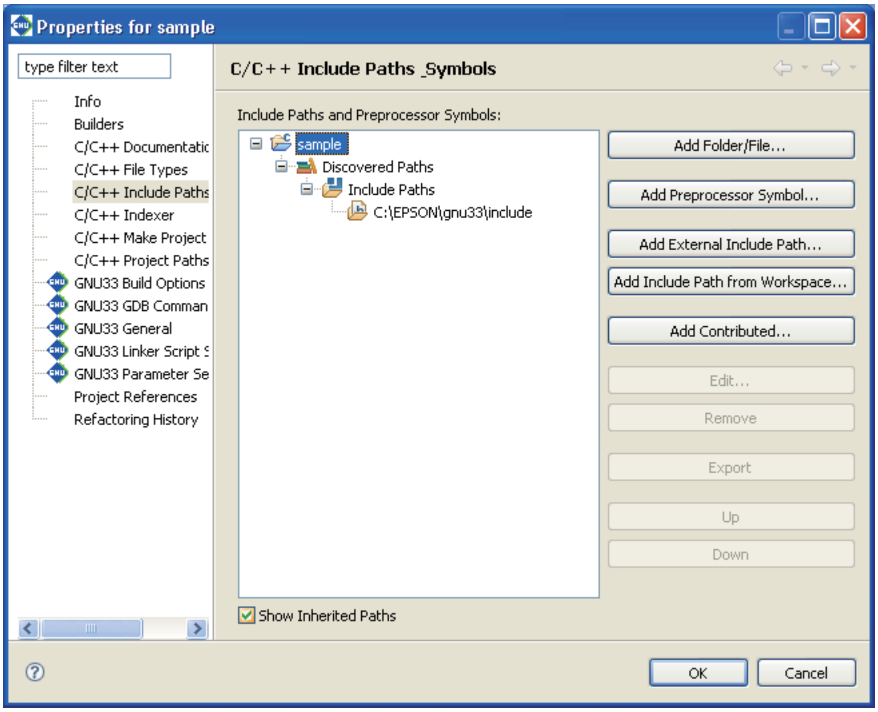
☐ Skip macro references (Search for macro references will not work)

Files to index up-front:

stdarg.h, stddef.h, sys/types.h

Make settings for the indexer used by C search and content assist. Do not change any of the settings on this page.

C/C++ Include Paths & Symbols

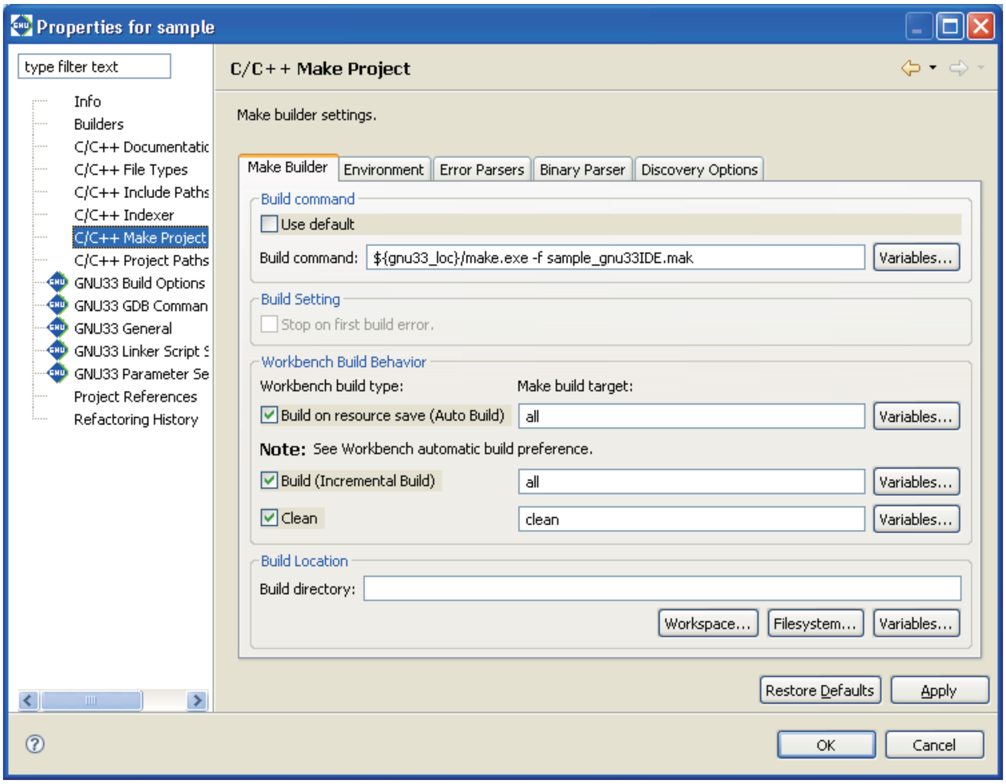

Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
C/C++ Include Paths _Symbols
Include Paths and Preprocessor Symbols:
sample
Discovered Paths
Include Paths
C:\EPSON\gnu33\include
Add Folder/File...
Add Preprocessor Symbol...
Add External Include Path...
Add Include Path from Workspace...
Add Contributed...
Edit...
Remove
Export
Up
Down
☑ Show Inherited Paths
OK
Cancel

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++ Make Project


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
Make Builder
Environment
Error Parsers
Binary Parser
Discovery Options
Variables...
Workspace...
Filesystem...
Restore Defaults
Apply
OK
Cancel

C/C++ Make Project

Make builder settings.

Build command

☐ Use default

Build command: ${gnu33_loc}/make.exe -f sample_gnu33IDE.mak

Build Setting

☐ Stop on first build error.

Workbench Build Behavior

Workbench build type: Make build target:

☑ Build on resource save (Auto Build) all

Note: See Workbench automatic build preference.

☑ Build (Incremental Build) all

☑ Clean clean

Build Location

Build directory:

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]

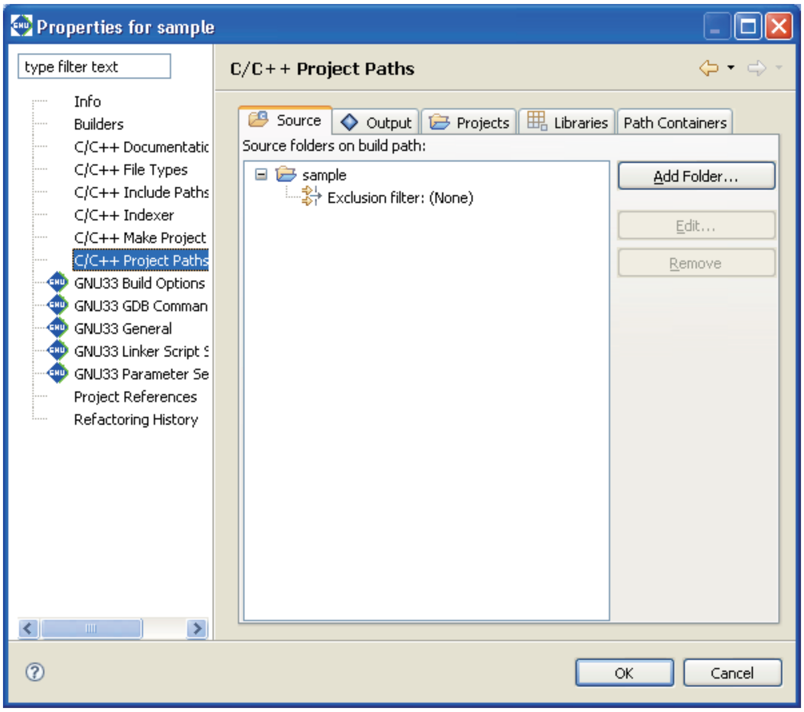
Although this field is used to specify the working directory during a build, no entry is required here.

If you are using your own makefiles, you must also edit or make changes on other pages. For more information, refer to Section 5.7.10, "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


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
C/C++ Project Paths
Source
Output
Projects
Libraries
Path Containers
Source folders on build path:
sample
Exclusion filter: (None)
Add Folder...
Edit...
Remove
OK
Cancel

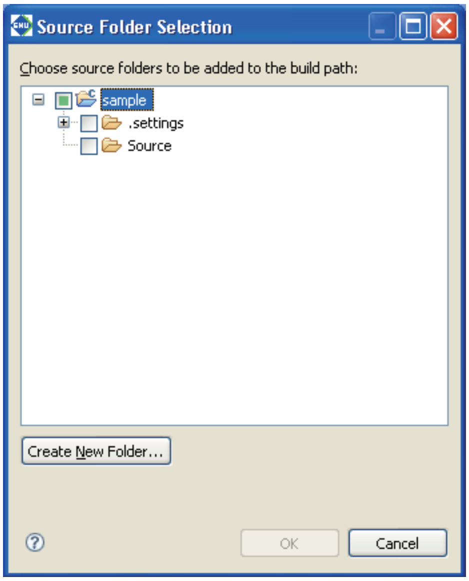
Set the source folder. The source files in the directory set here are written in the makefile used for a build.

[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.

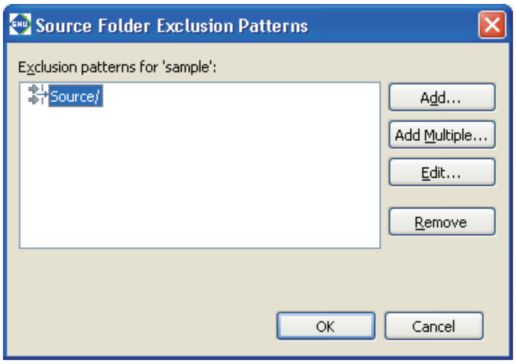

Source Folder Selection
Choose source folders to be added to the build path:
sample
.settings
Source
Create New Folder...
OK
Cancel

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/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.


Source Folder Exclusion Patterns
Exclusion patterns for 'sample':
Source/
Add...
Add Multiple...
Edit...
Remove
OK
Cancel

[Exclusion patterns for *<source directory>*:]

Lists the file name patterns by which to exclude certain C/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.

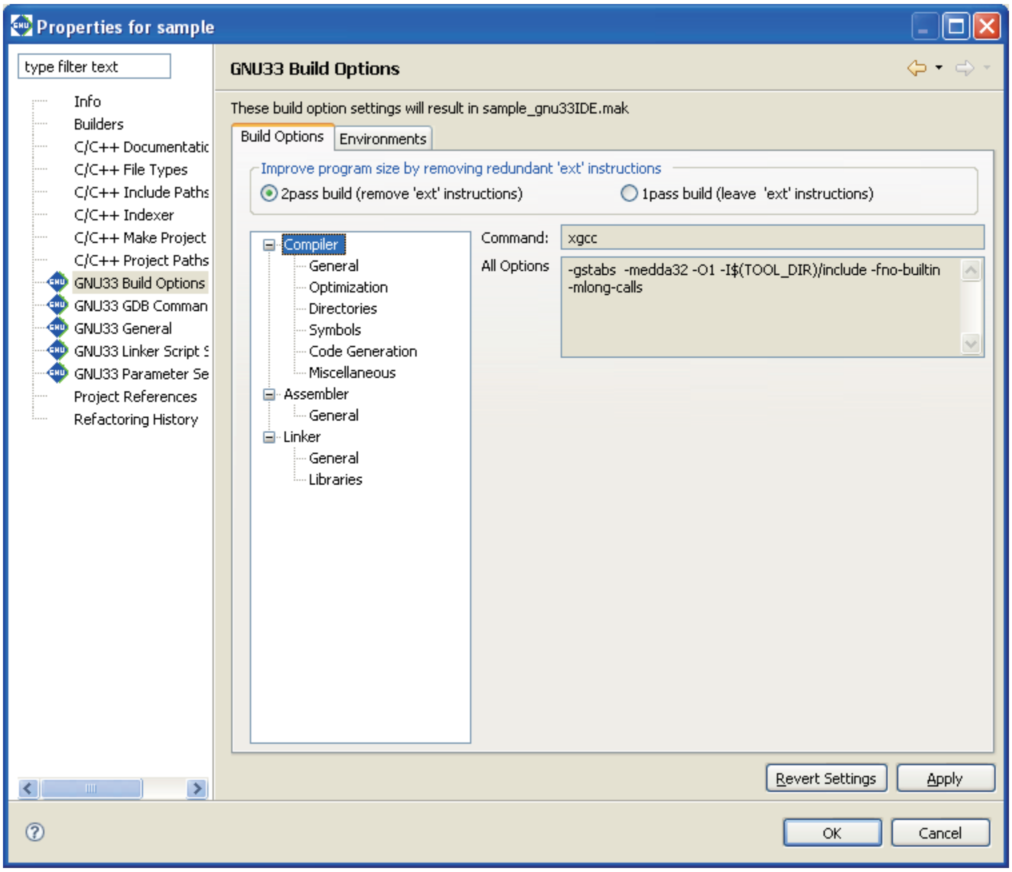
[Remove]

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.)

GNU33 Build Options


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

Command: xgcc

All Options: -gstabs -medda32 -O1 -I$(TOOL_DIR)/include -fno-builtin -mlong-calls

Set command line options for the compiler, assembler, and linker.

Furthermore, the radio buttons shown below are provided for selecting the two-pass make process (to optimize the extended instructions) or a normal one-pass make process.

[2pass build]

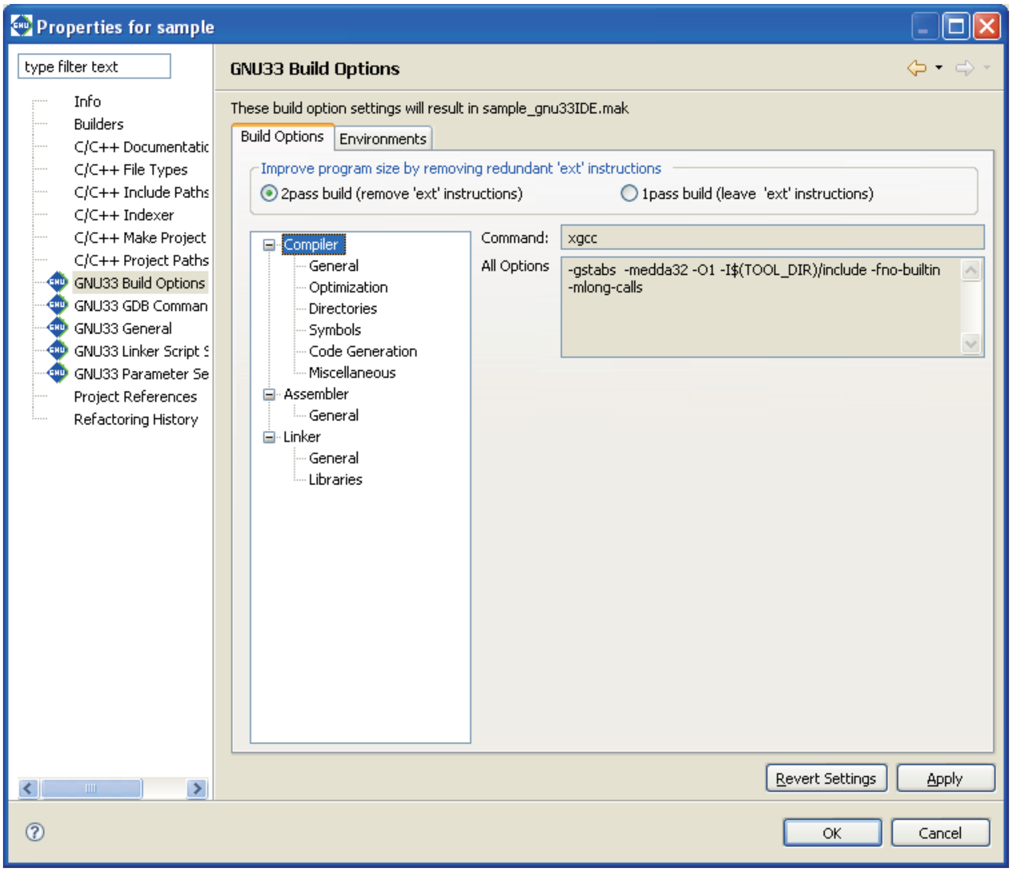
Selecting this button will generate a makefile that contains the two-pass make process to optimize the extended instructions. This increases the build time but reduces the program size.

[1pass build]

Selecting this button will generate a makefile that contains a one-pass make process that does not remove the ext instructions.

∗ If you click the [Apply] or [OK] button after settings in a [GNU33 Build Options] page have been changed, a dialog box appears for selecting "`clean`" build (see Section 5.7.9) to delete the files created with the previous settings (and rebuild).

[Build Options] tab > [Compiler]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

Command: xgcc

All Options: -gstabs -medda32 -O1 -I$(TOOL_DIR)/include -fno-builtin -mlong-calls

Shows the current settings for the compiler options.

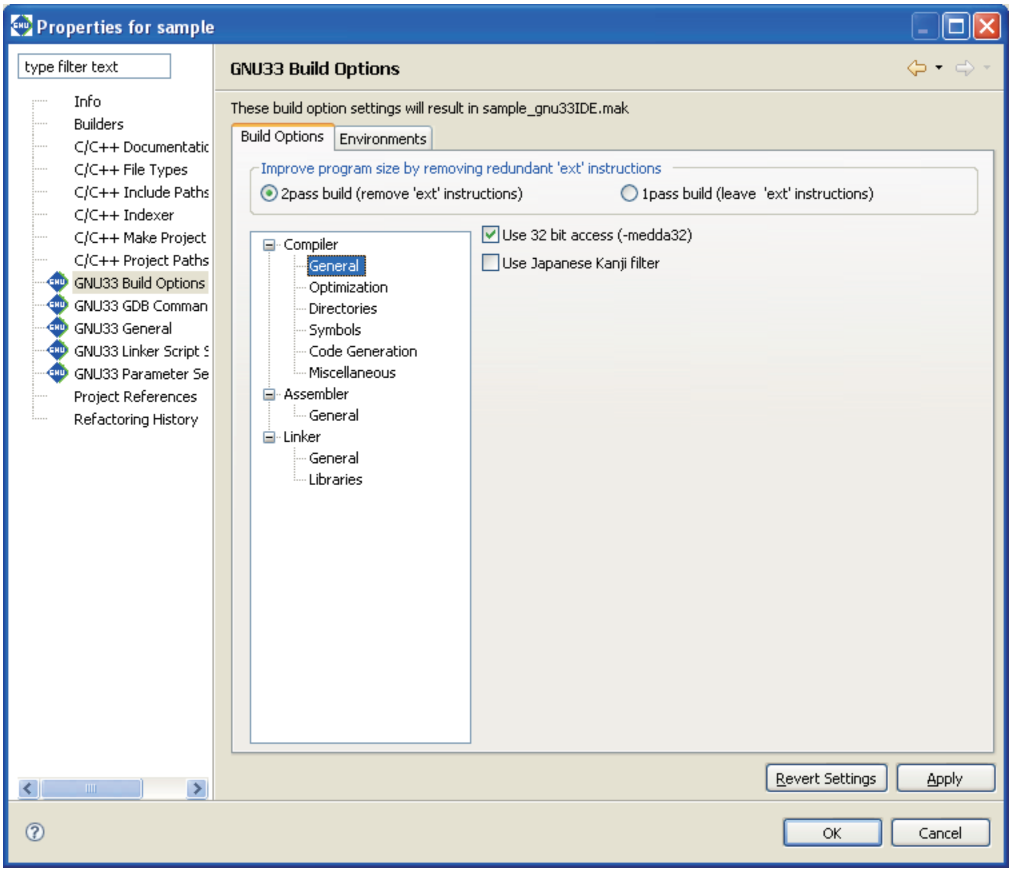
[Command:]

Shows the program name of the C/C++ compiler.

[All Options]

Shows the currently set compiler options.

[Build Options] tab > [Compiler] > [General]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions)   ○ 1pass build (leave 'ext' instructions)

☑ Use 32 bit access (-medda32)

☐ Use Japanese Kanji filter

Use this page to select the basic compiler options.

[Use 32 bit access (-medda32)]

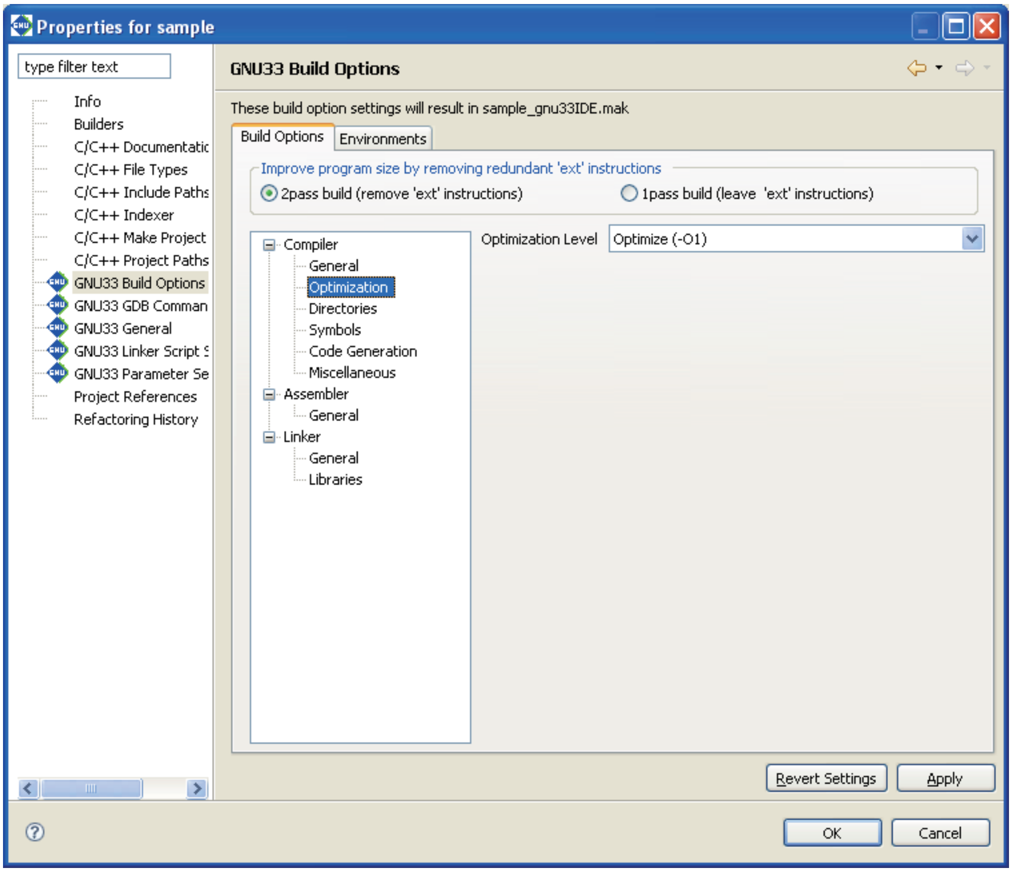
When this option is specified, the compiler outputs code for 32-bit access without using the default data area. You cannot specify this option in combination with the `-fPIC` option (in the [Code Generation] page).

[Use Japanese Kanji filter]

If this option is enabled, Shift JIS codes in the source will be read appropriately during compilation. Disabling the option is equivalent to specifying the `-mno-sjis-filt` option when calling a compiler, in which case the above processing is not performed.

The default status of the checkbox depends on the language of the OS used to run the **IDE**. The checkbox is selected by default if the **IDE** is launched in a Japanese-language OS environment; in other language versions, the checkbox is unselected by default. (When the checkbox is unselected, the `-mno-sjis-filt` option is specified during compilation.)

[Build Options] tab > [Compiler] > [Optimization]



Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

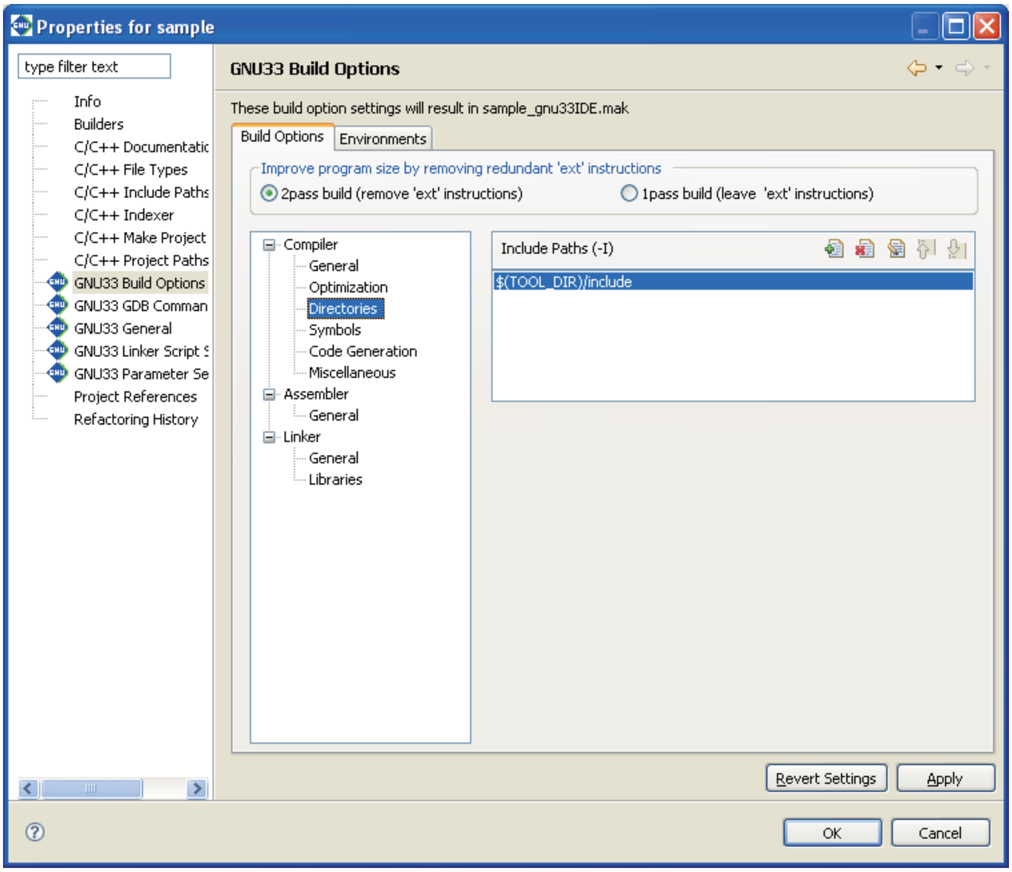
Optimization Level: Optimize (-O1)

Use this page to select compiler optimization options.

[Optimization Level]

Select the optimization level (`-O0` to `-O3`, `-Os`).

[Build Options] tab > [Compiler] > [Directories]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions
● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

Include Paths (-I)
$(TOOL_DIR)/include

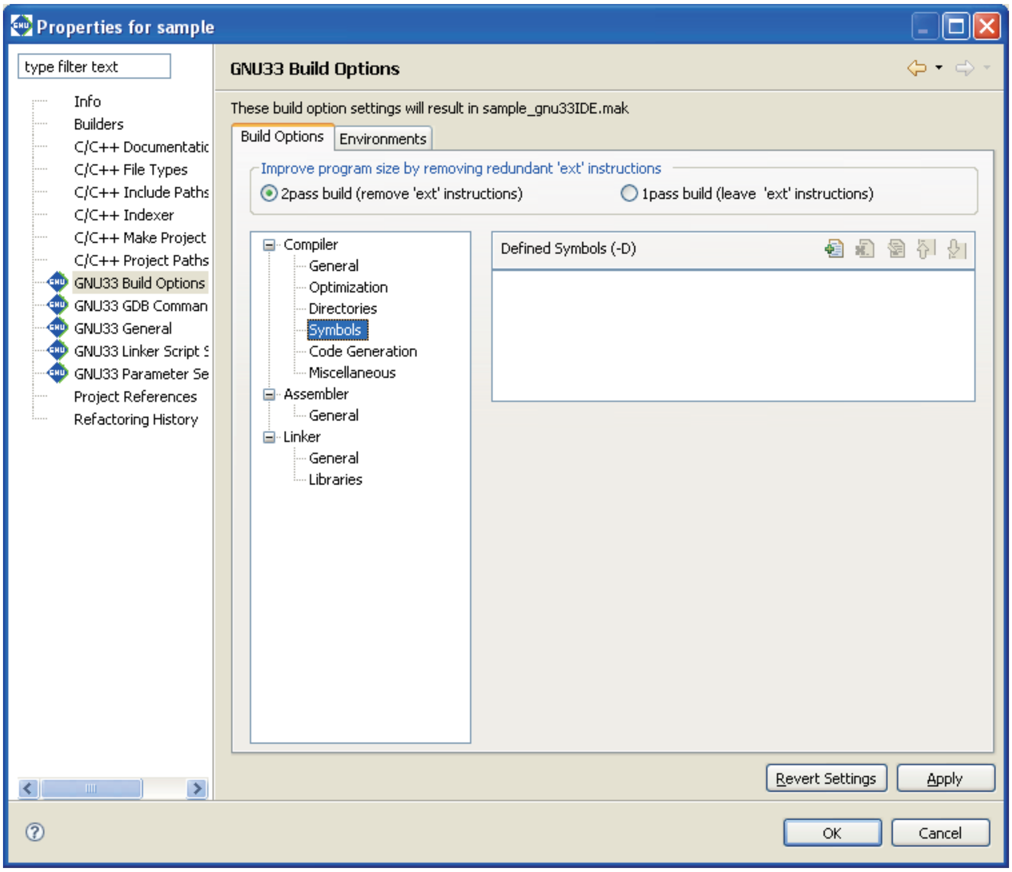
Use this page to set compiler search path options.

[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 > [Compiler] > [Symbols]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Defined Symbols (-D)
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions
● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

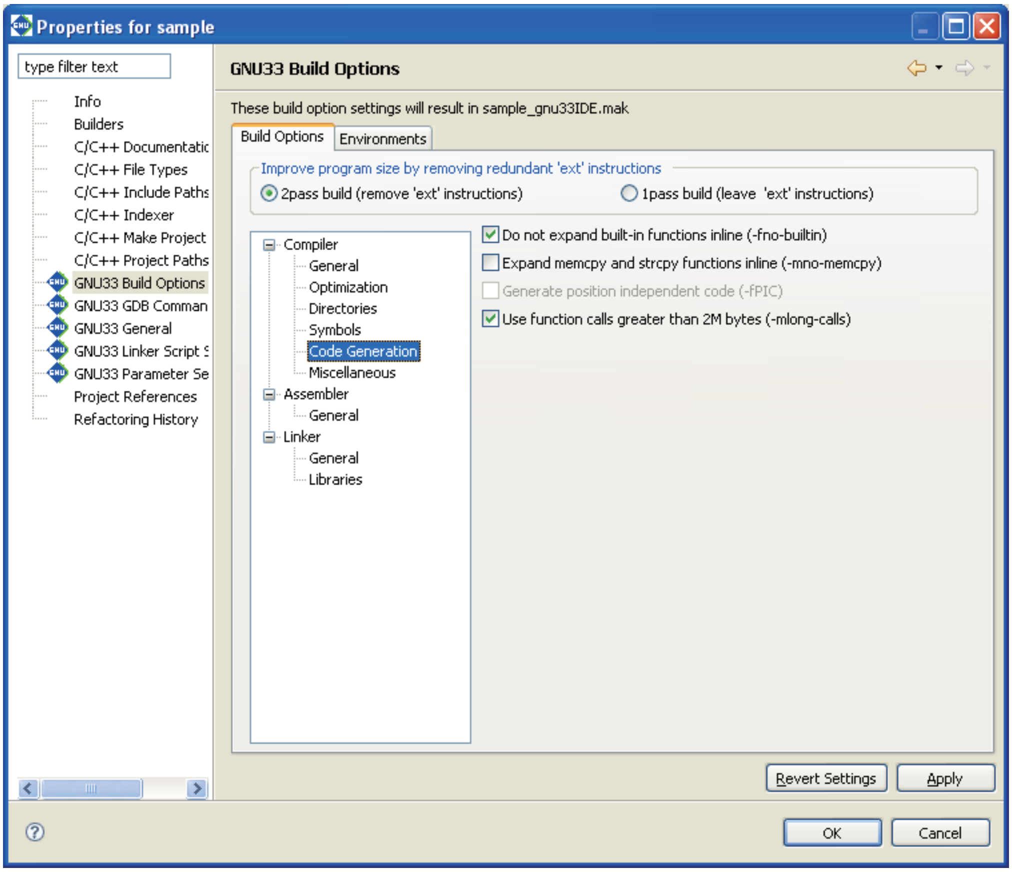
Use this page to set compiler macro-definition options.

[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 > [Compiler] > [Code Generation]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

☑ Do not expand built-in functions inline (-fno-builtin)

☐ Expand memcpy and strcpy functions inline (-mno-memcpy)

☐ Generate position independent code (-fPIC)

☑ Use function calls greater than 2M bytes (-mlong-calls)

Use this page to select compiler code-generation options.

[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".

[Expand memcpy and strcpy functions inline (-mno-memcpy)]

When this option is specified, the `memcpy()` and `strcpy()` functions are expanded in-line.

[Generate position independent code (-fPIC)]

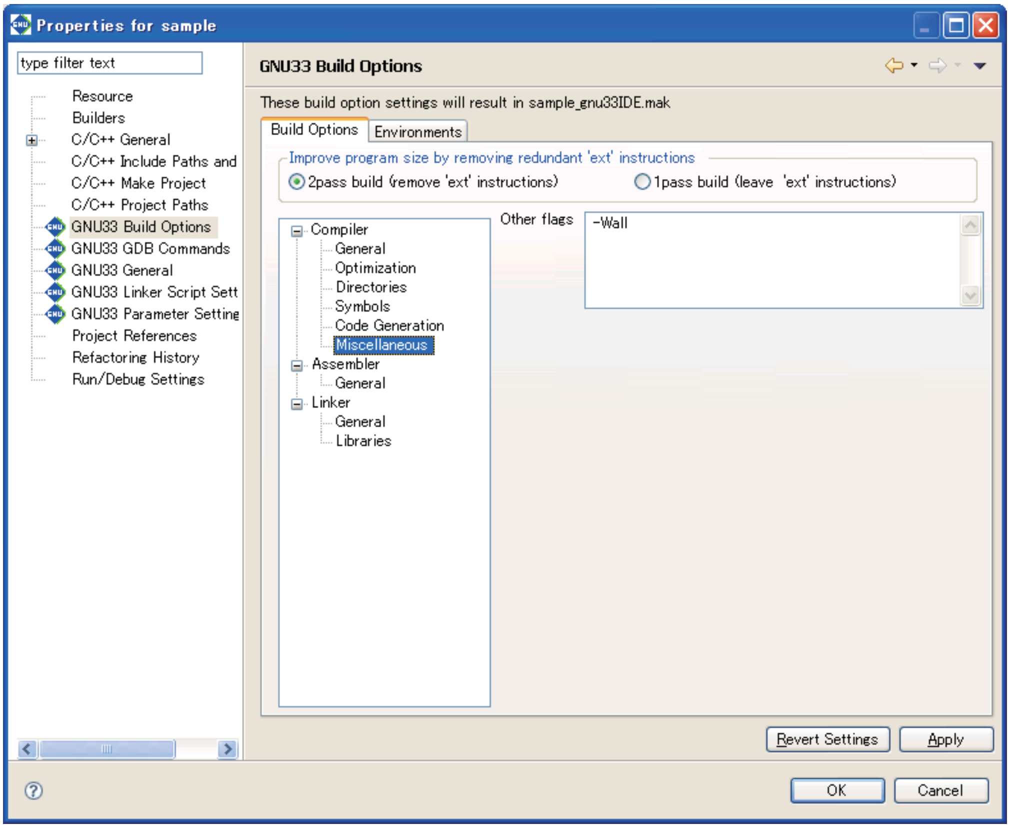
Specifying this option generates position-independent code.

This option cannot be specified in combination with the `-medda32` option (in the [General] page).

[Use function calls greater than 2M bytes (-mlong-calls)]

Function calls of 2M bytes or more from the current position are executed.

[Build Options] tab > [Compiler] > [Miscellaneous]


Properties for sample
type filter text
Resource
Builders
C/C++ General
C/C++ Include Paths and
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Commands
GNU33 General
GNU33 Linker Script Sett
GNU33 Parameter Setting
Project References
Refactoring History
Run/Debug Settings
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions)   ○ 1pass build (leave 'ext' instructions)

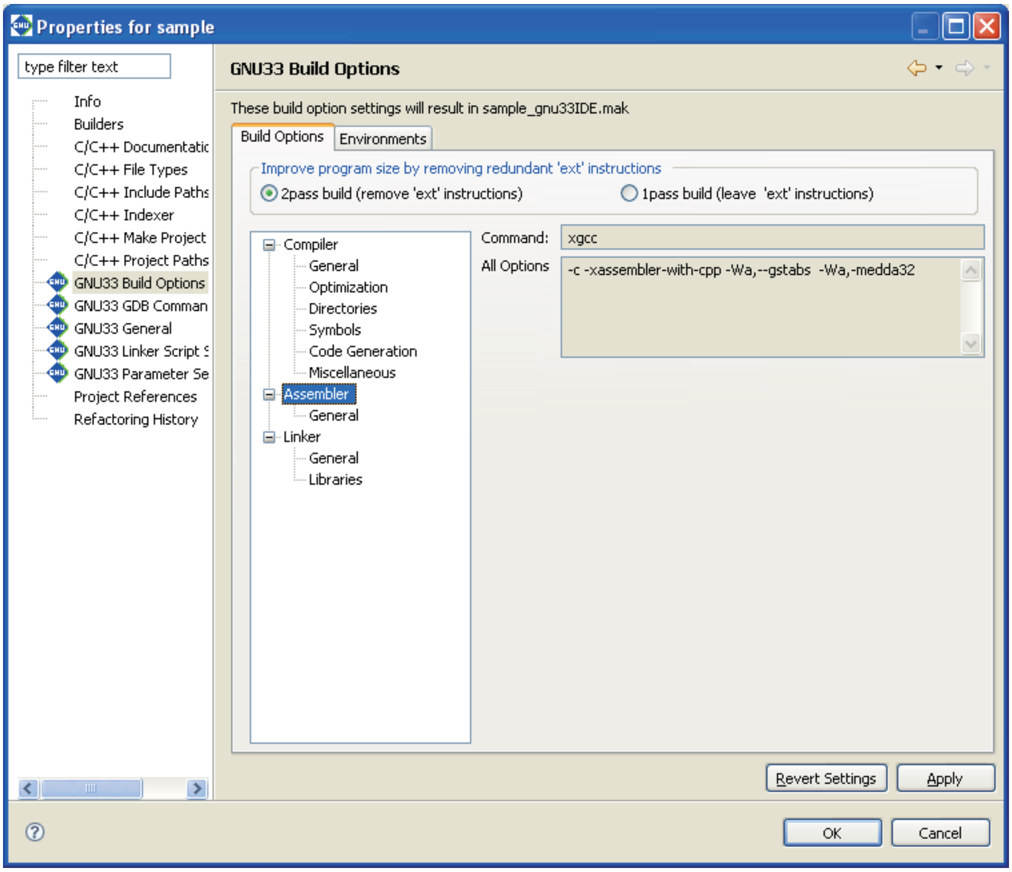
Other flags: -Wall

Use this page to set other compiler options.

[Other flags] (default: -Wall)

Enter other options directly into this text field. Insert one or more spaces between each option.
If the "S1C33401" is selected for CPU type, the `-mc33adv` and `-mc33401` options are also specified.
Similarly, if the "S1C33PE" is selected for CPU type, the `-mc33pe` option is specified.
When the two-pass make process is selected, the `-S` option is automatically specified for compilation and an assembly file (extension: .ext0) is output, and this file is assembled by the assembler.
It is not necessary to specify the compiler `-S` option or compiler `-C` option in [Other flags].
The assembled results of the C source can be confirmed in the < C SOURCE FILE NAME.ext0 > file.
Also the `-gstabs` option is automatically specified.

[Build Options] tab > [Assembler]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

Command: xgcc

All Options: -c -xassembler-with-cpp -Wa,--gstabs -Wa,-medda32

Shows the current settings for the assembler options.

[Command:]

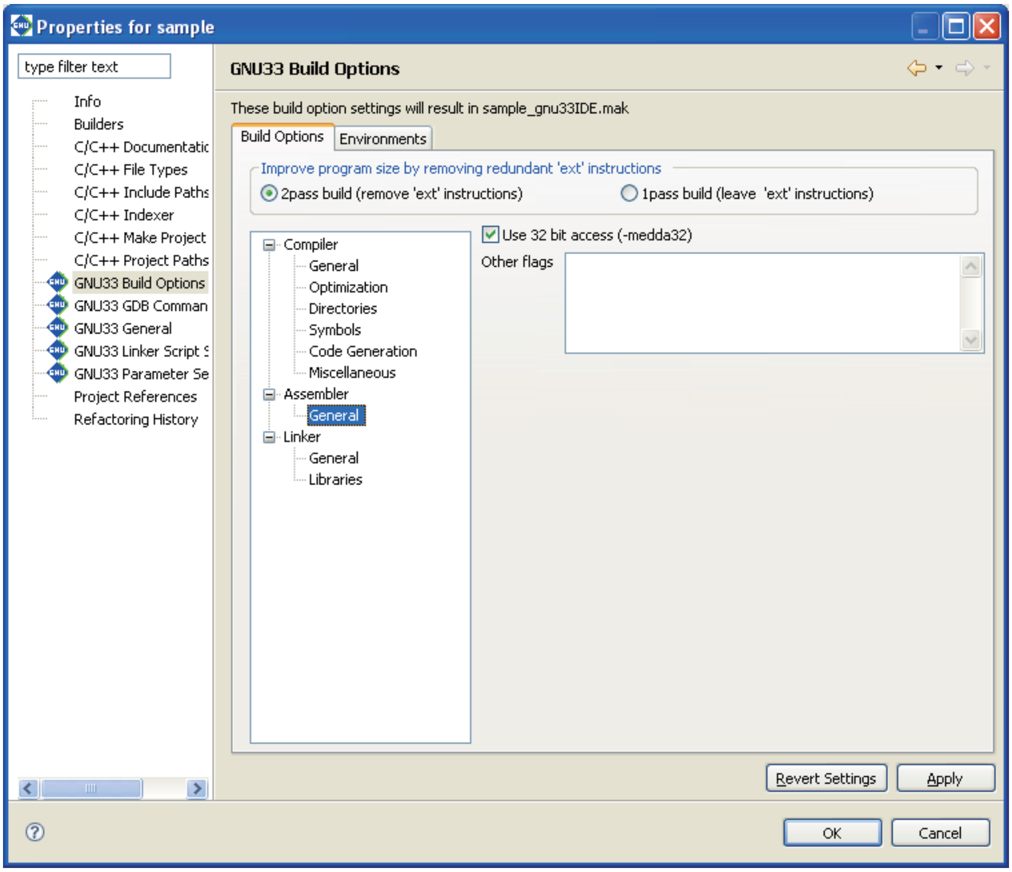
Shows the program name of the compiler*.

[All Options]

Shows the currently set options.

Note: When the `-c -xassembler-with-cpp` option is specified, the **IDE** assembles the assembler source using the specified C compiler. It is not necessary to specify the compiler `-c` option or `-xassembler-with-cpp` option to All Options.

[Build Options] tab > [Assembler] > [General]



Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

☑ Use 32 bit access (-medda32)

Other flags

Use this page to select assembly options. Note that the `-c`, `-xassembler-with-cpp`, and `-Wa, --gstabs` options are always added, regardless of the settings made below.

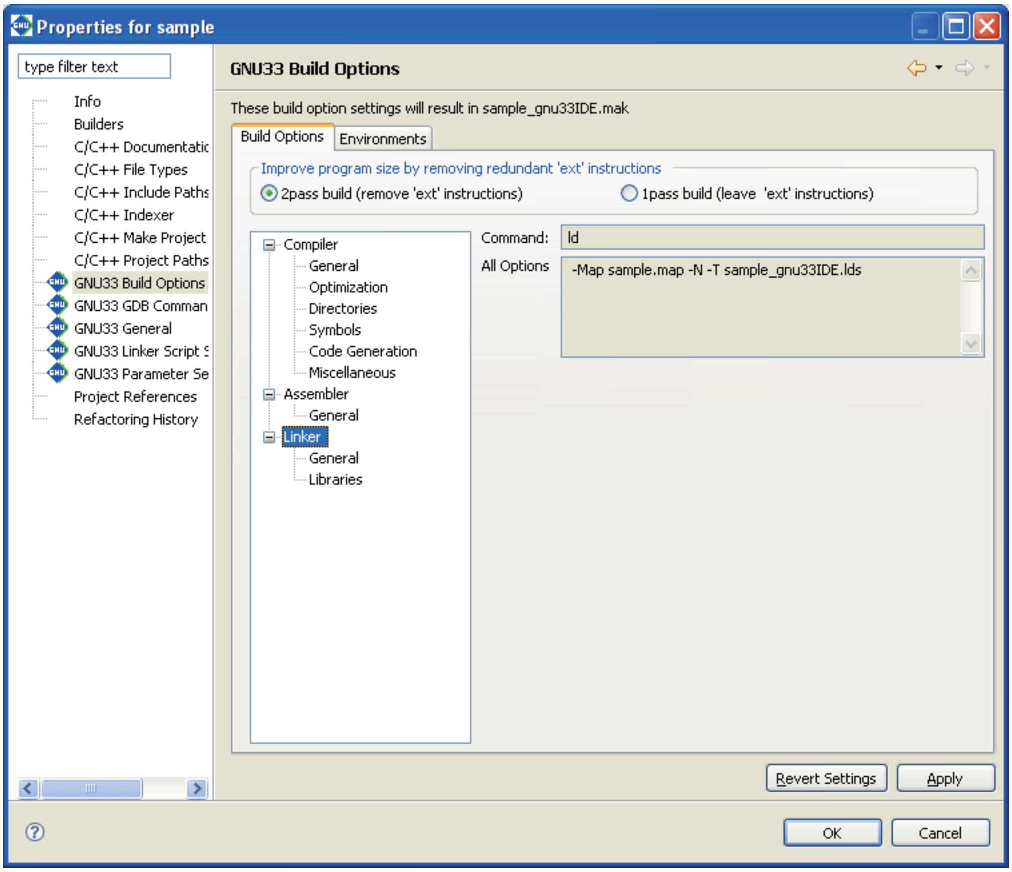
[Use 32 bit access (-medda32)]

When this option is specified, the assembler outputs code for 32-bit access without using the default data area.

[Other flags]

Enter the options to be passed to the assembler. Insert one or more spaces between each option.
If the "SC33401" is selected for CPU type, the `-mc33adv` and `-mc33401` options are also specified.
Similarly, if the "S1C33PE" is selected for CPU type, the `-mc33pe` option is specified.
The options entered are passed to the assembler as "`-Wa,`<*option*>`,` ...".

[Build Options] tab > [Linker]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

Command: ld

All Options: -Map sample.map -N -T sample_gnu33IDE.lds

Shows the current settings for the linker options.

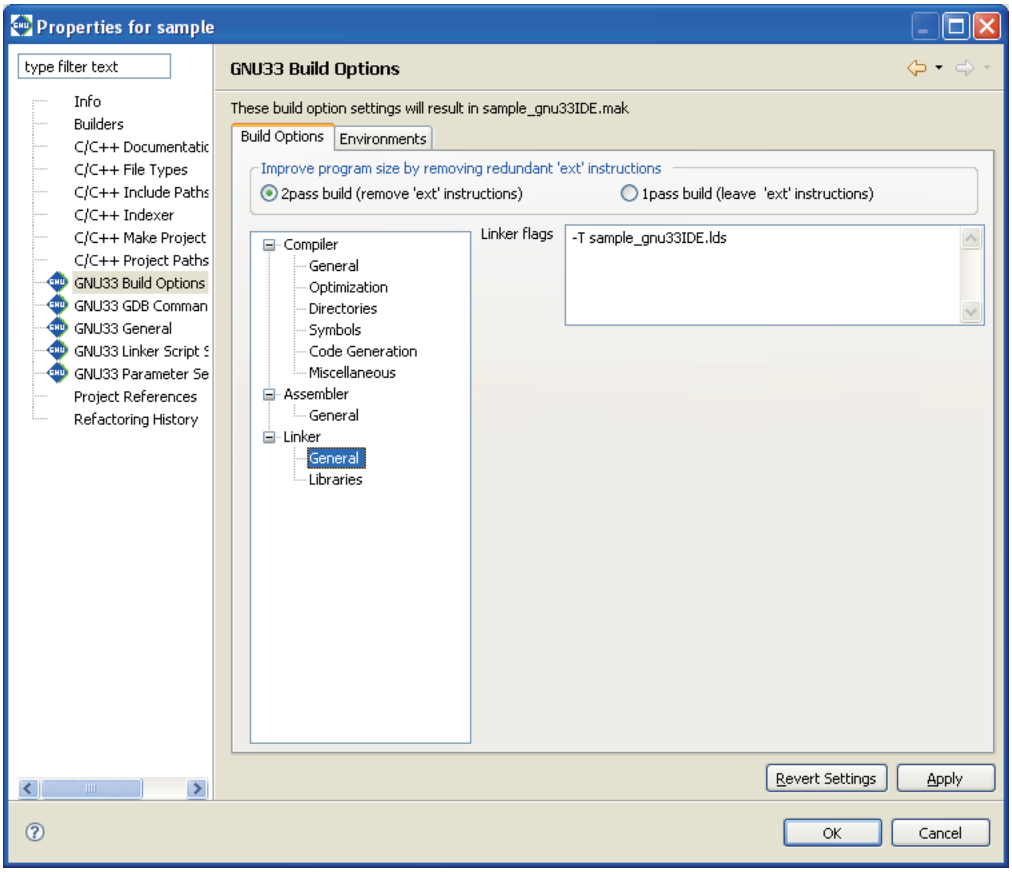
[Command:]

Shows the program name of the linker.

[All Options]

Shows the currently set options.

[Build Options] tab > [Linker] > [General]



Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions)   ○ 1pass build (leave 'ext' instructions)

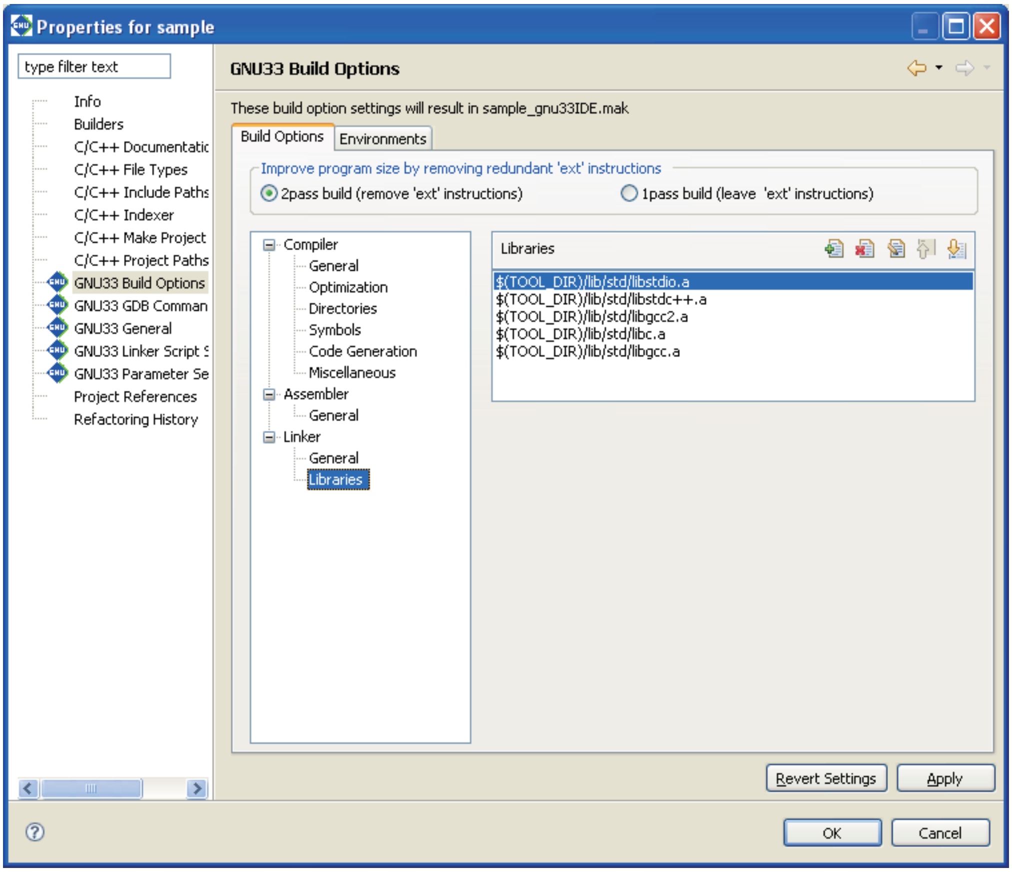
Linker flags: -T sample_gnu33IDE.lds

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 > [Linker] > [Libraries]


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
Compiler
General
Optimization
Directories
Symbols
Code Generation
Miscellaneous
Assembler
General
Linker
General
Libraries
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Improve program size by removing redundant 'ext' instructions

● 2pass build (remove 'ext' instructions) ○ 1pass build (leave 'ext' instructions)

Libraries

```
$(TOOL_DIR)/lib/std/libstdio.a
$(TOOL_DIR)/lib/std/libstdc++.a
$(TOOL_DIR)/lib/std/libgcc2.a
$(TOOL_DIR)/lib/std/libc.a
$(TOOL_DIR)/lib/std/libgcc.a
```

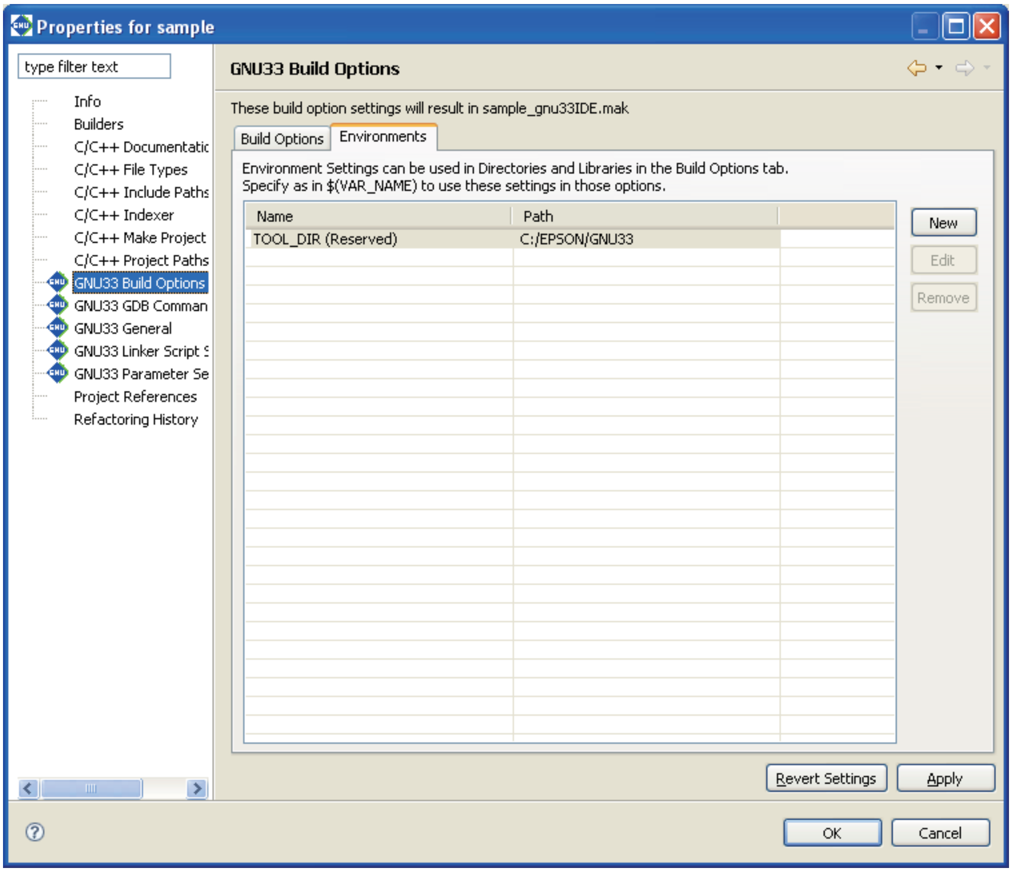
Use this page to set the libraries to be linked.

[Libraries] (default: `libstdio.a`, `libstdc++.a`, `libgcc2.a`, `libc.a`, `libgcc.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.

[Environments] tab


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Build Options
Build Options
Environments
New
Edit
Remove
Revert Settings
Apply
OK
Cancel

These build option settings will result in sample_gnu33IDE.mak

Environment Settings can be used in Directories and Libraries in the Build Options tab.
Specify as in $(VAR_NAME) to use these settings in those options.

Name	Path
TOOL_DIR (Reserved)	C:/EPSON/GNU33

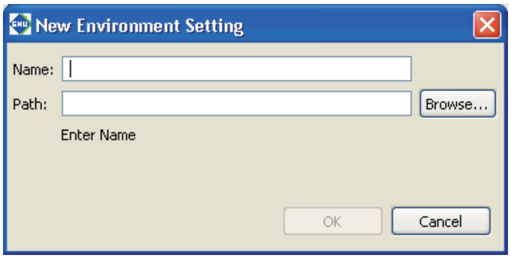
Use this page to manage environment variables used for specifying paths in the [Compiler] > [Directories] and [Linker] > [Libraries] pages in the [Build Options] tab.

Environment variable list

Lists the available environment variables and the paths defined. `TOOL_DIR` is the reserved variable in which the gnu33 tool directory is defined, and cannot be edited or removed.

[New]

Displays a dialog box to define a new environment variable.
Enter a variable name in the [Name:] text box, and the path to be defined in the [Path:] text box on the [New Environment Setting] dialog box that appears by clicking this button, then click [OK]. The path may be selected in the dialog box that appears by clicking the [Browse...] button.


New Environment Setting
Name:
Path:
Browse...
OK
Cancel

Enter Name

[Edit]

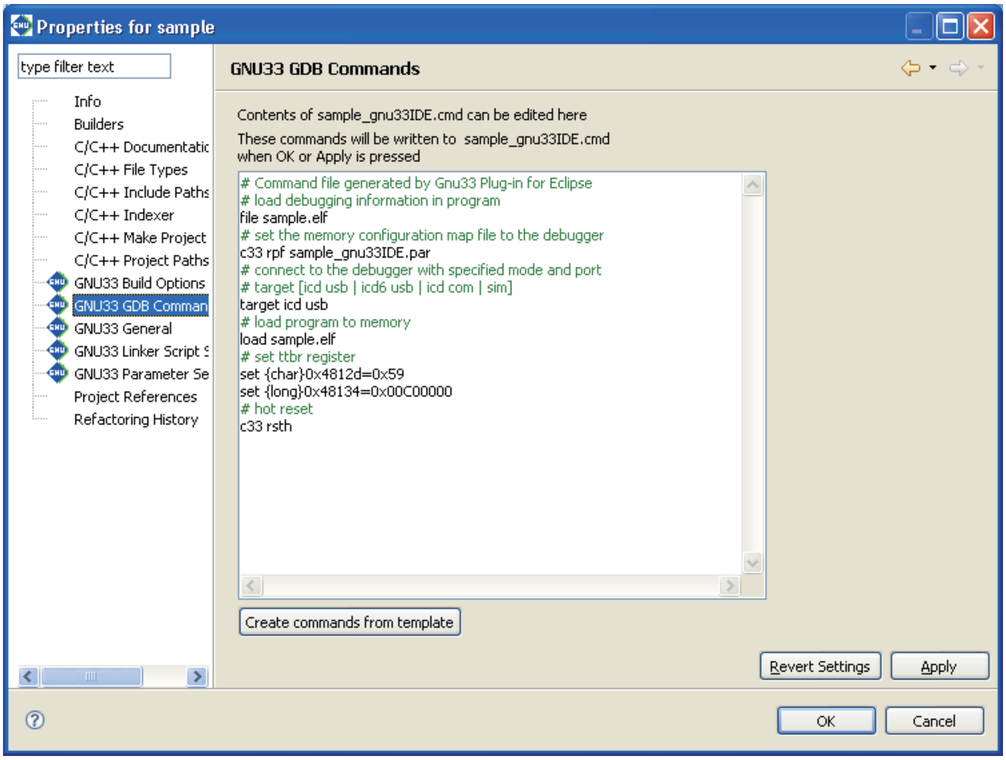
Displays a dialog box similar to that appeared by the [New] button to edit the name and path definition of the environment variable that has been selected in the list.

[Remove]

Removes the environment variable selected in the list.

Notes:
- A path to be entered in the [New Environment Setting] dialog box cannot include characters other than single-byte alphanumeric characters, '_', ':', '/' and '\' or '\'. Japanese and other two-byte characters cannot be used.
- Use the defined environment variables as `$(`*environment variable*`)` format. If an environment variable is described in another format, it will not be replaced and an error will occur during a build process.

GNU33 GDB Commands


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 GDB Commands
Create commands from template
Revert Settings
Apply
OK
Cancel

Contents of sample_gnu33IDE.cmd can be edited here

These commands will be written to sample_gnu33IDE.cmd
when OK or Apply is pressed

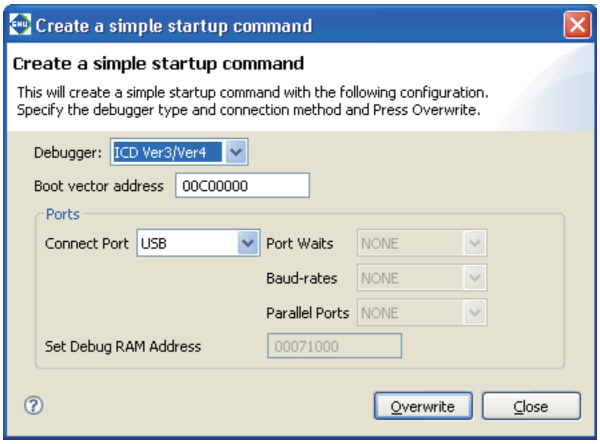
```
# Command file generated by Gnu33 Plug-in for Eclipse
# load debugging information in program
file sample.elf
# set the memory configuration map file to the debugger
c33 rpf sample_gnu33IDE.par
# connect to the debugger with specified mode and port
# target [icd usb | icd6 usb | icd com | sim]
target icd usb
# load program to memory
load sample.elf
# set ttbr register
set {char}0x4812d=0x59
set {long}0x48134=0x00C00000
# hot reset
c33 rsth
```

This page is used to edit the debugger start up command file for this project. The contents displayed here are written exactly to the command file.

[Create commands from template]

Displays the [Create a simple startup command] dialog box shown below to set the connect mode and other debugging conditions. The command file contents shown on this page reflect the settings in the [Create a simple startup command] dialog box.

Create a simple startup command


Create a simple startup command
Create a simple startup command
Ports
Overwrite
Close

This will create a simple startup command with the following configuration.
Specify the debugger type and connection method and Press Overwrite.

Debugger: ICD Ver3/Ver4

Boot vector address 00C00000

Connect Port USB

Port Waits NONE

Baud-rates NONE

Parallel Ports NONE

Set Debug RAM Address 00071000

Set the connect mode and the port to which the ICD is connected.

[Debugger:]

Select the debugger (connect mode) to connect to. You will see five choices, but the selectable options will change according to the target CPU selected in the [GNU33 General] page.

ICD Ver6	When using ICD Ver. 6 (S5U1C33001H1400) to debug
ICD Ver3/Ver4	When using ICD Ver. 3 (S5U1C33001H1100) or ICD Ver. 4 (S5U1C33001H1200) to debug
ICD Ver2	When using ICD Ver. 2 (S5U1C33000H) to debug
MON	When using the debug monitor (S5U1C330M1D1 + S5U1C330M2S) to debug
Simulator	To debug with a PC only.

Table 5.10.1.1 Target CPUs and connect modes

Mode	C33 STD Core	C33 PE Core	S1C33401
ICD Ver6	○	○	○
ICD Ver3/Ver4	○	○	○
ICD Ver2	○	×	×
MON	○	×	×
Simulator	○	○	○

[Boot vector address]

Enter the boot vector address in hexadecimal notation (omit 0x). If the CPU type is C33 STD or C33 PE Core, the address must be specified in 400KB increments. For the S1CC33401, the address may be specified in 1KB increments.

The **IDE** generates a command file that includes a command for setting the boot vector address with this value. The value appearing here by default is the address that was specified when the project was created.

[Ports]

Set the desired COM port if you selected ICD Ver2 or MON for [Debugger:] above. If you selected ICD Ver3/Ver4, ICD Ver6 or Simulator, the port is set to USB and NONE, respectively, and no settings are required.

[Connect Port]

Select the COM port (COM1–COM8) to connect with the S5U1C33000H (ICD Ver.2) or S5U1C330M1D1 (MON).

[Port Waits]

Specify the duration of the wait state to be inserted when the COM port on the PC side is opened. Set a value ranging from 0 to 20 seconds. Some PCs (such as laptops) may not be able to initiate communications normally unless a wait state is inserted after the COM port is opened.

[Baud-rates]

Select the baud rate for the COM port: 115200 bps or 38400 bps. If you selected MON for [Debugger:], you must select 115200 bps.

[Parallel Ports]

Specify a port number to select the parallel port for transferring programs to the target. Specify NONE, LPT1, or LPT2, which correspond to no parallel ports used, parallel port 1, and parallel port 2, respectively. This selection is enabled only if you selected ICD Ver2 for [Debugger:].

[Set Debug RAM Address]

Set the memory address on the target side to be used by the debugger during a debug process. This setting is available only if you selected S1C33PE for the target CPU in the [GNU33 General] page and ICD Ver3/Ver4 or ICD Ver6 for [Debugger:] above. The [Set Debug RAM Address] command is valid only for this combination of parameter selections.

Make sure the RAM area specified here is not used in a program.

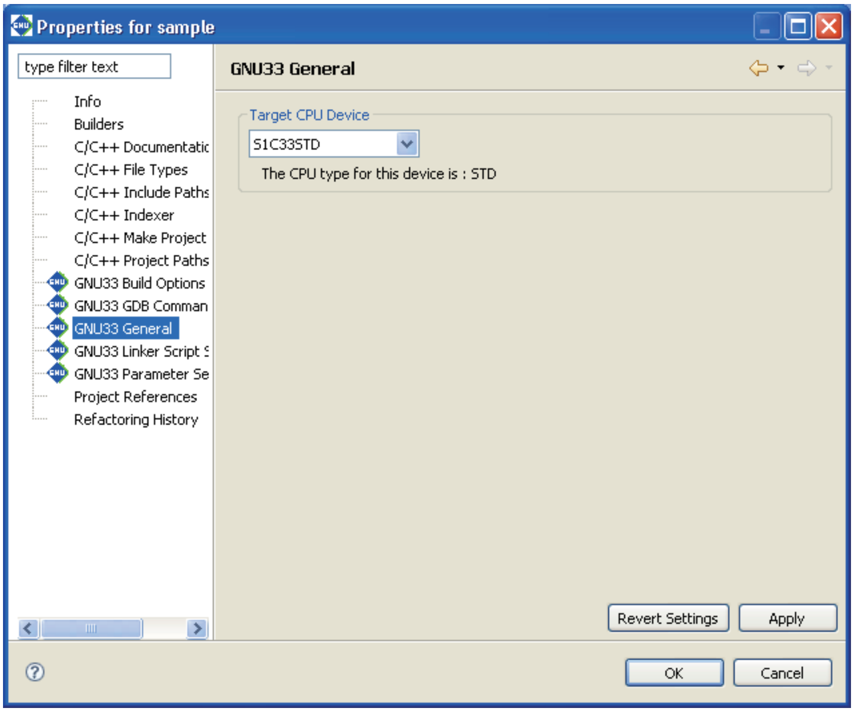
[Overwrite]

Applies the above settings to the command file shown in the [GNU33 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 [GNU33 GDB Commands] page does not reflect the contents changed in the dialog box.

GNU33 General


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
Revert Settings
Apply
OK
Cancel

GNU33 General

Target CPU Device

S1C33STD

The CPU type for this device is : STD

Select the target processor.

[Target CPU Device]

Select a target processor from the following three options:

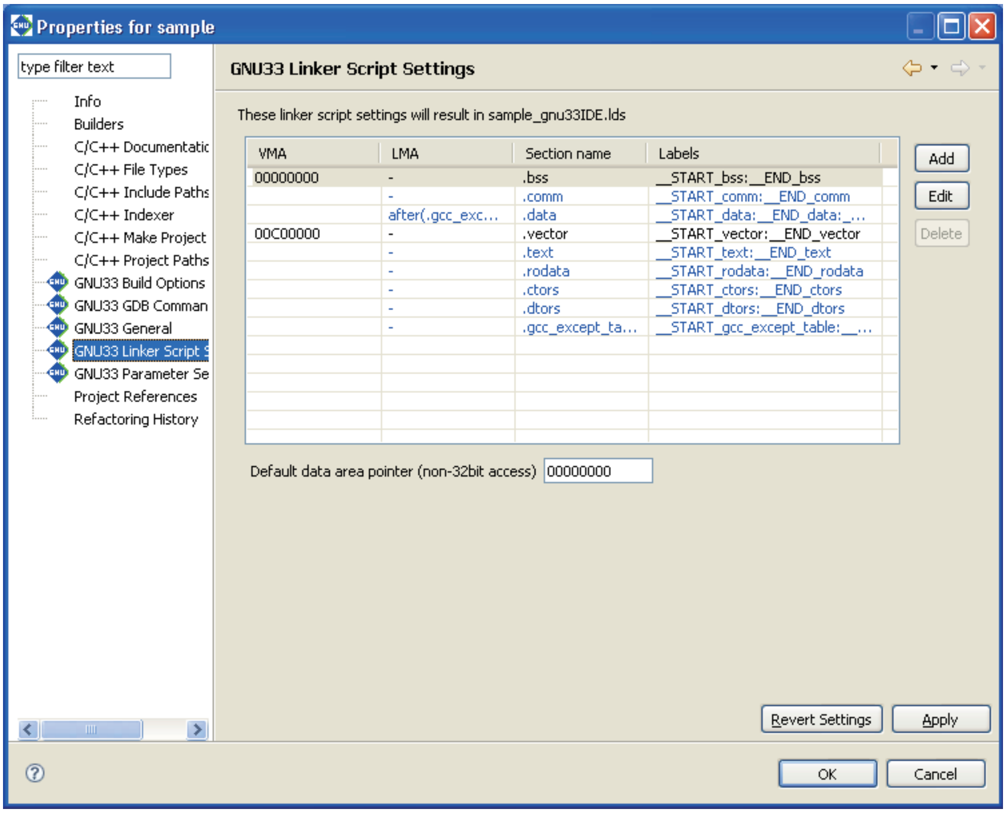
S1C33STD: When the target processor is a device with the C33 STD Core embedded

S1C33PE: When the target processor is a device with the C33 PE Core embedded

S1C33401: When the target processor is the S1C33401

* 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 5.7.9) to delete the files created with the previous settings (and rebuild).

GNU33 Linker Script Settings



Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Linker Script Settings
Add
Edit
Delete
Revert Settings
Apply
OK
Cancel

These linker script settings will result in sample_gnu33IDE.lds

VMA	LMA	Section name	Labels
00000000	-	.bss	__START_bss:__END_bss
	-	.comm	__START_comm:__END_comm
	after(.gcc_exc...	.data	__START_data:__END_data:_...
00C00000	-	.vector	__START_vector:__END_vector
	-	.text	__START_text:__END_text
	-	.rodata	__START_rodata:__END_rodata
	-	.ctors	__START_ctors:__END_ctors
	-	.dtors	__START_dtors:__END_dtors
	-	.gcc_except_ta...	__START_gcc_except_table:__...

Default data area pointer (non-32bit access) 00000000

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 are 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 that 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]

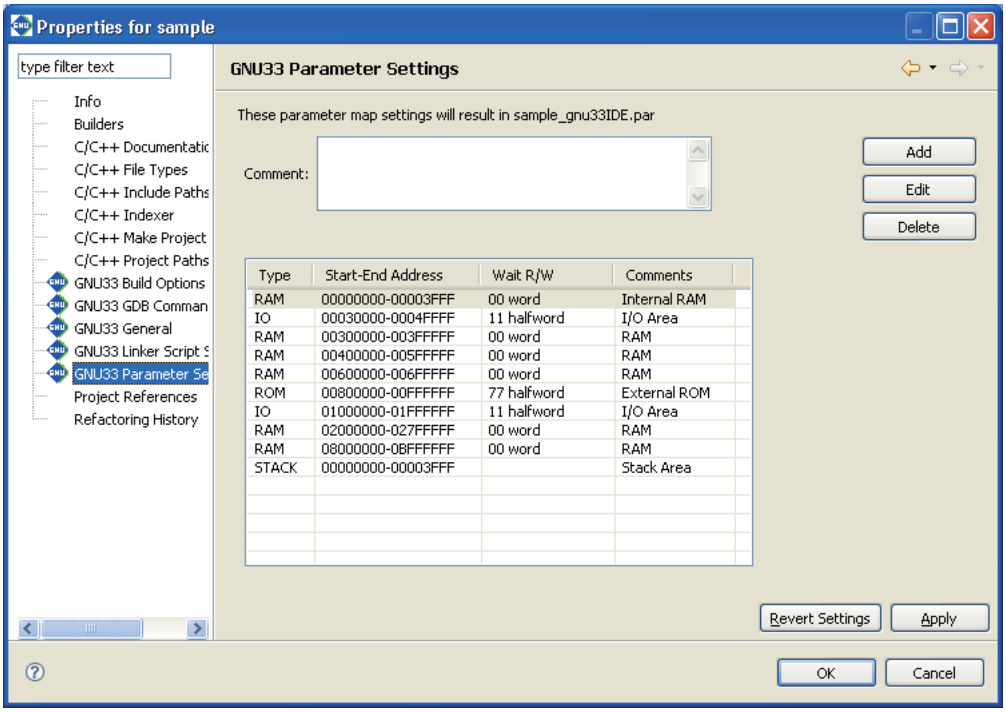
Deletes the section information selected in the list.

[Default data area pointer]

Set the default data area pointer (address) to be written in a linker script.

For more information on how to edit a linker script, refer to Section 5.7.7, "Editing a Linker Script".

GNU33 Parameter Settings


Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
GNU33 Parameter Settings
Comment:
Add
Edit
Delete
Revert Settings
Apply
OK
Cancel

These parameter map settings will result in sample_gnu33IDE.par

Type	Start-End Address	Wait R/W	Comments
RAM	00000000-00003FFF	00 word	Internal RAM
IO	00030000-0004FFFF	11 halfword	I/O Area
RAM	00300000-003FFFFF	00 word	RAM
RAM	00400000-005FFFFF	00 word	RAM
RAM	00600000-006FFFFF	00 word	RAM
ROM	00800000-00FFFFFF	77 halfword	External ROM
IO	01000000-01FFFFFF	11 halfword	I/O Area
RAM	02000000-027FFFFF	00 word	RAM
RAM	08000000-0BFFFFFF	00 word	RAM
STACK	00000000-00003FFF		Stack Area

Edit the parameter file used in the debugger.

[Comment:]

You can enter any comments (up to 255 characters) here. The comments entered here are written to the parameter file.

Area list

Shows information for one area per line. Information is listed in alphabetical order, except that all stack area information is displayed at the bottom of the list.

[Type]

Shows the area type (ROM, RAM, IO, or STACK).

[Start-End Address]

Shows the start and the end addresses of the area in hexadecimal notation.

[Wait R/W]

The first two-digit value shows 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", "halfword", and "word" indicate the access size by 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 indicate the start of a comment.

[Add]

Adds area information.

[Edit]

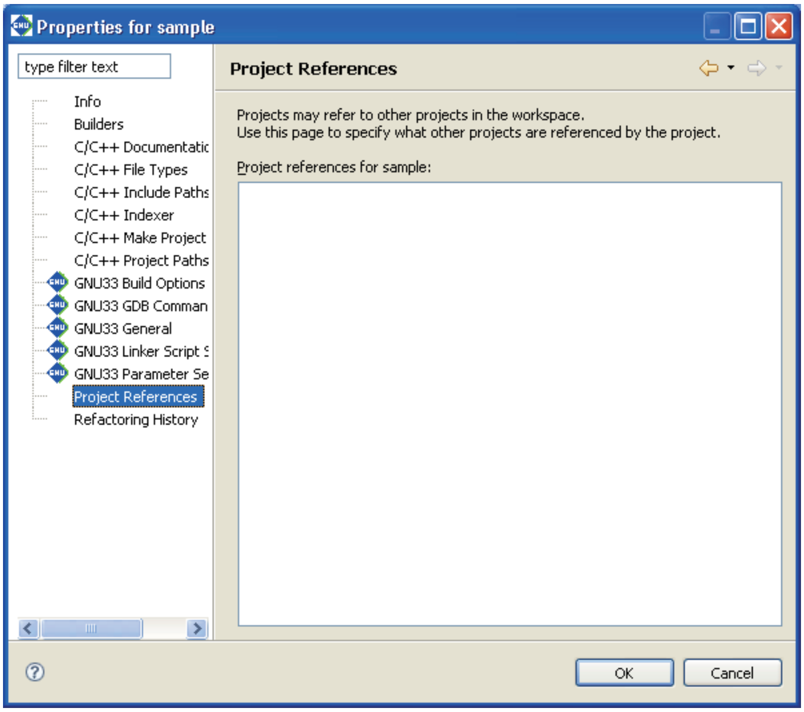
Edits the area information selected in the list.

[Delete]

Deletes the area information selected in the list.

For more information on how to edit a parameter file, refer to Section 5.8.1, "Generating a Parameter File".

Project References



Properties for sample
type filter text
Info
Builders
C/C++ Documentatic
C/C++ File Types
C/C++ Include Paths
C/C++ Indexer
C/C++ Make Project
C/C++ Project Paths
GNU33 Build Options
GNU33 GDB Comman
GNU33 General
GNU33 Linker Script S
GNU33 Parameter Se
Project References
Refactoring History
Project References
OK
Cancel

Projects may refer to other projects in the workspace.
Use this page to specify what other projects are referenced by the project.

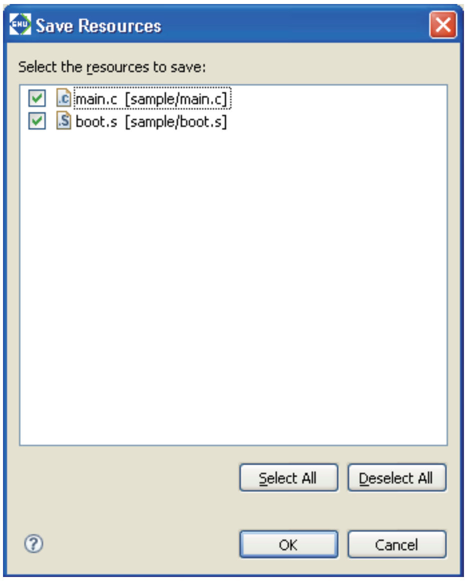
Project references for sample:

Select the project to be referenced.

[Project references for sample:]

Select the other projects to be referenced by the current project.

5.10.2 Save Resources



Select the resources to save:

☑ main.c [sample/main.c]
☑ boot.s [sample/boot.s]

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.

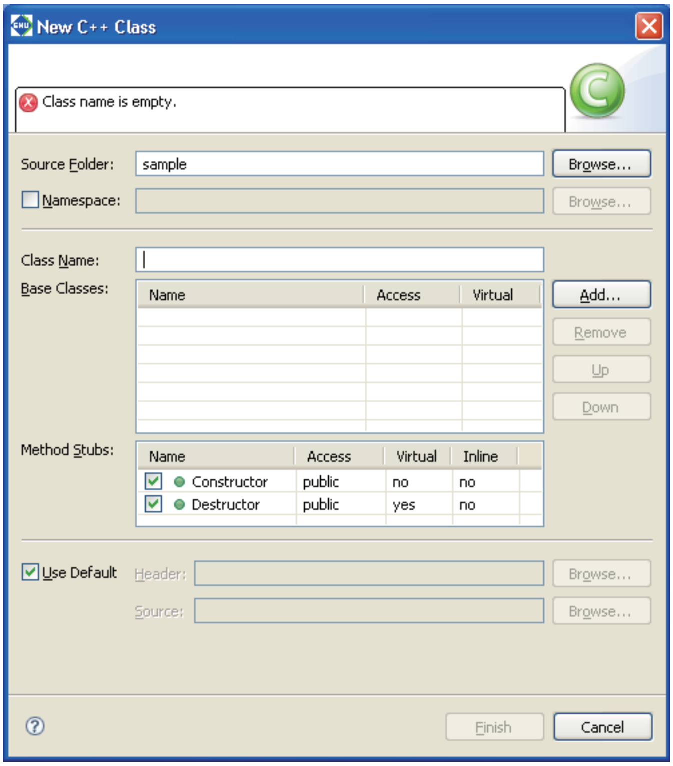
[OK]

Closes the documents in the editor after saving the selected files.

[Cancel]

Cancels the action invoking the dialog box. The documents are neither saved nor closed.

5.10.3 New C++ Class



Class name is empty.

Source Folder: sample

☐ Namespace:

Class Name:

Base Classes:

Name	Access	Virtual

Method Stubs:

Name	Access	Virtual	Inline
☑ Constructor	public	no	no
☑ Destructor	public	yes	no

☑ Use Default Header: Source:

This dialog box is displayed when you select [New] > [Other...] > [C++] > [Class] from the [File] menu (or click the [New C++ Class] button in the toolbar) to create a new class.

[Source Folder:]

Enter the name of directory in which you want to locate the source and header file to be created or select it using the [Browse...] button.

[Namespace:]

When defining the class belong to a specific namespace, select the check box and enter the name to the text box or select it using the [Browse...] button.

[Class Name:]

Enter the name of the new class you want to create.

[Base Classes:]

If you are creating a derivative class from an already created class in the current project, enter the name of the base class from which you want to inherit to the [Base class:] list. You can select an already defined class from the [Choose Base Class] dialog box displayed when you click the [Add...] button.

Example:

Class declaration where AClass is specified for the base class and a derivative class named BClass is created from the base class (the selected access specifier is public)

```
class BClass : public AClass{
public:

BClass();
      virtual ~BClass();
};
```

Unnecessary class name in this list can be removed using the [Remove] button. The order in the list can also be edited using the [Up] and [Down] buttons.

Note: To select a base class or a namespace, [Fast C/C++ Indexer] must be selected in the [C/C++ Indexer] page of the project property in advance.

[Method Stubs:]

Select the check box to define constructor and/or destructor.

[Access]

Select an access modifier to be written in the constructor and/or destructor from "`public`", "`protected`" or "`private`".

[Virtual]

When "yes" is selected, a virtual destructor is declared; when "no" is selected, a non-virtual destructor is declared.

[Inline]

When "yes" is selected, constructor and destructor definitions are included in the header file, and only an include statement for the header file is written to the source file. When "no" is selected, the source file includes constructor and destructor definitions.

[Use Default]

When creating a new header and source files, select the check box. When appending the definitions to existing files, deselect the check box and enter the file name or select it using the [Browse...] button.

[Finish]

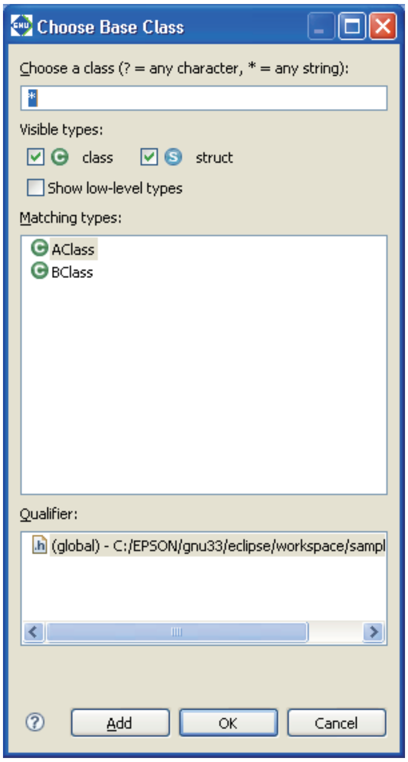
Generates a new class.

[Cancel]

Cancels creation of a new class.

5.10.4 Choose Base Class

Note: To display the [Choose Base Class] dialog box for selecting a base class, [Fast C/C++ Indexer] must be selected on the [C/C++ Indexer] page of the project property in advance.



Choose a class (? = any character, * = any string): *

Visible types:

☑ class ☑ struct

☐ Show low-level types

Matching types:

- AClass
- BClass

Qualifier:

(global) - C:/EPSON/gnu33/eclipse/workspace/sampl

This dialog box is displayed when you click the [Add...] button to select a base class for [Base Classes:] in the [New C++ Class] dialog box.

[Choose a class (? = any character, * = any string):]

Enter a string to restrict to a subset of the classes listed in [Matching types:]. Use "?" as a single-character wildcard and "*" as a string wildcard.

Example:

?Class	The first character may be any character. (e.g., AClass)
*Class	Class may be preceded by any character. (e.g., GetTimeClass)

[Visible types:]

Select the type to be listed in [Matching types:].

[class]

Shows classes.

[struct]

Shows structures.

[Show low-level types]

Shows low-level types.

[Matching types:]

Lists the types defined in the current project that meets the above conditions. Select your desired base class here.

[Qualifier:]

Shows qualifiers.

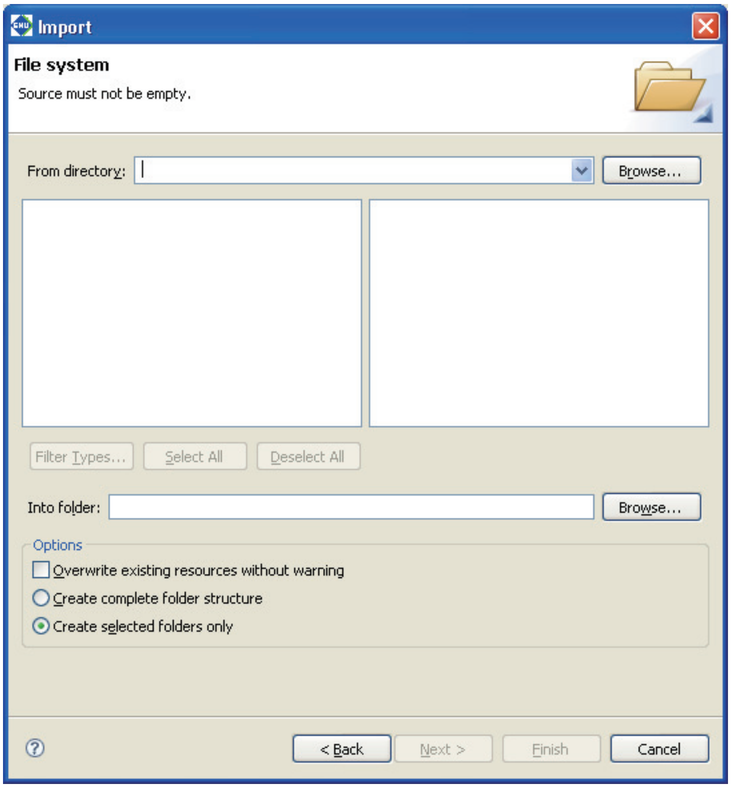
[OK]

Inputs the base class selected in [Matching types:] into the [Base Classes:] text field in the [New C++ Class] dialog box.

[Cancel]

Cancels the selection.

5.10.5 Import > File system



File system

Source must not be empty.

☐ Overwrite existing resources without warning

○ Create complete folder structure

● Create selected folders only

This dialog box is displayed if you select [General] > [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.

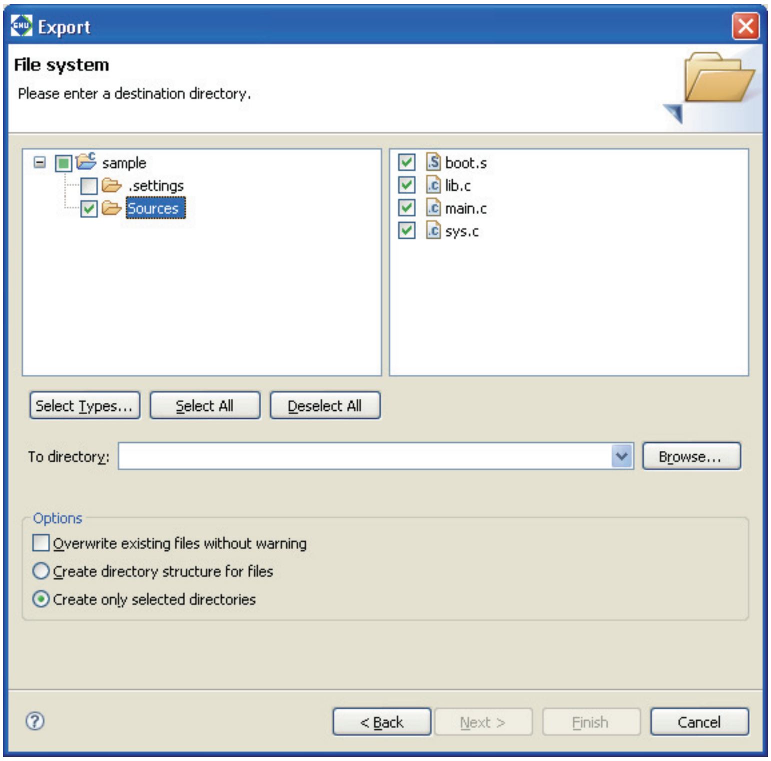
[Create complete folder structure]

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.

5.10.6 Export > File system


Export
File system
Please enter a destination directory.
sample
.settings
Sources
boot.s
lib.c
main.c
sys.c
Select Types...
Select All
Deselect All
To directory:
Browse...
Options
Overwrite existing files without warning
Create directory structure for files
Create only selected directories
< Back
Next >
Finish
Cancel

This dialog box appears if you select [General] > [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.

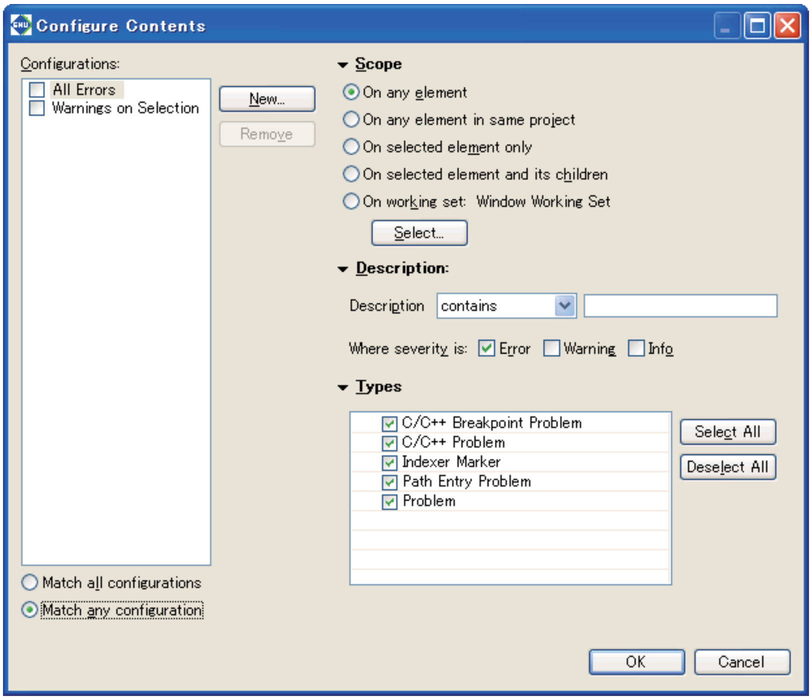
[Create directory structure for files]

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.

5.10.7 Filters


Configure Contents
Configurations:
☐ All Errors
☐ Warnings on Selection
New...
Remove
○ Match all configurations
● Match any configuration
Scope
● On any element
○ On any element in same project
○ On selected element only
○ On selected element and its children
○ On working set: Window Working Set
Select...
Description:
Description contains
Where severity is: ☑ Error ☐ Warning ☐ Info
Types
☑ C/C++ Breakpoint Problem
☑ C/C++ Problem
☑ Indexer Marker
☑ Path Entry Problem
☑ Problem
Select All
Deselect All
OK
Cancel

To display this dialog box, select [Configure Contents] on the View menu (▽) of the [Problems], [Bookmarks], or [Tasks] view. 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.

[Configurations:]

Displays a list of filters that have been set. Use each checkbox to select whether to apply the filter.
To modify filter conditions, select a filter name from this list and select the corresponding conditions.
If two or more filters are selected, all filter conditions will be ORed.

[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:].

[Completed] ([Tasks] view only)

[Completed]	Displays only completed tasks.
[Not Completed]	Displays only incomplete tasks.

[Priority] ([Tasks] view only)

Select to restrict the display based on task priority level. Specify the priority of items to be displayed by selecting the desired checkbox (High, Normal, or Low).

[Description]

contains	Only items containing the string entered in the text box will be displayed in [Description].
doesn't contain	Only items that do not contain the string entered in the text box will be displayed in [Description].

To disable this restriction, leave this text box blank.

[Where severity is:] ([Problems] view only)

Select to restrict the display based on error criticality. Specify the display content (Error, Warning, or Info) by selecting the desired checkbox.

[Types] ([Problems] view and [Tasks] view only)

Select the type of list to be displayed to a specific view.

[Select All]	Selects all items displayed in [Types].
[Deselect All]	Deselects all items displayed in [Types].

[OK]

Begins filtering with the set conditions.

[Cancel]

Cancels filter 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
.cproject	IDE project file (CDT)	Prohibited	Required
.cdtproject	IDE project file (CDT) Project created prior to GNU33 v3.2.1	Prohibited	Required
.gnu33project	IDE project file (GNU33)	Prohibited	Required
GDB33 Launch for *<project name>*.launch	GDB launch setting file	Prohibited	Required
*<project name>*_gnu33IDE.cmd	GDB command file	Possible ∗1	Required
\.settings	Project settings directory	Prohibited	Required
\.externalToolBuilders	Project settings directory (builder)	Prohibited	Required
*<project name>*_gnu33IDE.mak	makefile	Prohibited	Not required
*<project name>*_gnu33IDE.lds	Linker script file	Prohibited	Not required
*<project name>*_gnu33IDE.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/C++ Compiler

This chapter explains how to use the **xgcc** C/C++ compiler, and provides details on interfacing with the assembly source. For information about the standard functions of the C/C++ compiler and the syntax of the C/C++ source programs, refer to the ANSI C/C++ literature generally available on the market.

6.1 Functions

The **xgcc** C/C++ compiler compiles C/C++ source files to generate an assembly source file that includes C33 Core instruction set mnemonics, extended instructions, and assembler directives. The **xgcc** is a gnu C/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 S1C33 Family.
Furthermore, since this C/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/C++ compiler consists of four files: **xgcc.exe**, **cpp.exe**, **cc1.exe**, and **cc1plus.exe**.

The **xgcc** is based on the C/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.
This C/C++ compiler has passed the compiler evaluation test of Japan Novel Corporation.
Of bugs that were detected in the compiler evaluation test, those necessitating restrictions on the C/C++ compiler are listed in "\gnu33\doc\release_history.pdf" and Section 6.11, "Known Issues". For details, refer to the above file or section.

6.2 Input/Output Files

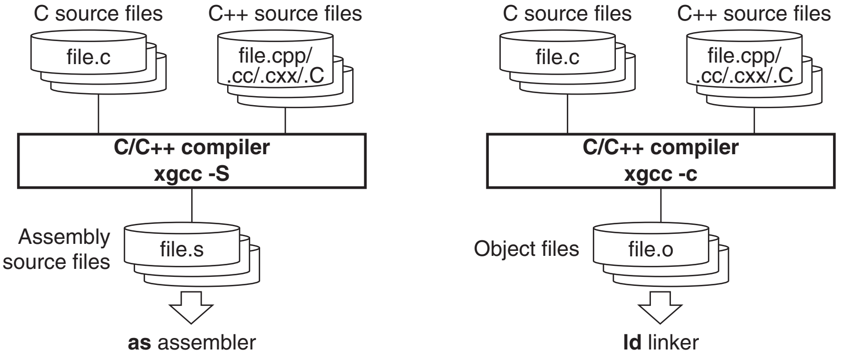


Figure 6.2.1 Flowchart

6.2.1 Input Files

C source file

File format: Text file
File name: <*file name*>`.c`
Description: File in which the C source program is described.

C++ source file

File format: Text file
File name: <*file name*>`.cpp`, <*file name*>`.cc`, <*file name*>`.cxx`, <*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/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/C++ compiler, use the command shown below.

`xgcc` ***<options>*** ***<file name>***

<*options*>	See Section 6.3.2.
<*file name*>	Specify C/C++ source file name(s) including the extension (`.c`, `.cpp`, `.cc`, `.cxx`, `.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/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/C++ compiler. Do not specify the `-S` or `-E` option simultaneously when this option is used.
Default:	The **xgcc** C/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/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/C++ compiler generates the elf executable object file.

`-E`

Function:	**Execute C preprocessor only**
Description:	When this option is specified, the **xgcc** C/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/C++ compiler generates the elf executable object file.

`-B`*<directory>*

Function: **Specify compiler search path**

Description: This option is used to add the <*directory*> to the search paths of the **xgcc** C/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/cc1plus**, 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/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/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/C++ compiler searches include files in the current directory and the C_INCLUDE_PATH directory.

`-fno-builtin`

Function: **Disables 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`, `alloca`, `cos`, `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.

`-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 used from any source file.

Macro name	Generation conditions
`__c33`	Always generated regardless of core type.
`__c33std`	Generated if the `-mc33adv` and `-mc33pe` options are not specified.
`__c33adv`	Generated if the `-mc33adv` option is specified.
`__c33pe`	Generated if the `-mc33pe` option is specified.

`-O0`, `-O`, `-O2`, `-O3`, `-Os`

Function: **Optimization**

Description: Specify one of the five switches to perform optimized processing.

The compiler generates a speed-oriented or size-oriented optimized code according to the specified switch. The `-O2` and `-O3` switches specify optimization to increase the execution speed, and the `-Os` switch specifies optimization to reduce the code size. Unless a switch is specified, code generation is not optimized. The greater the value of `-O`, the higher the functionality of optimization, with the cost that some debugging information may not be output or other problems may arise. If code generation cannot be executed normally, reduce the value of the optimization option. Register interlocks are disregarded during the optimization phase. Since `-O2`/`-O3` are provided for speed-priority optimization, the code size may be larger than for `-O`.

Normally, `-O` should be specified. The basic make file generated by the **IDE** specifies `-O` option when invoking the **xgcc** C/C++ compiler.

Described below are the features of each option.

`-O0`

Does not optimize.

Storage is reserved for the stack even when an unused local variable is declared. The code is compiled directly as is, so that even unnecessary code such as one that assigns a value to a local variable that is not referenced from anywhere is generated directly. Variable values loaded into registers are not reused. However, local variables declared as `register` are optimized, so that any unnecessary local variables are deleted.

`-O`/`-O1`

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`.

Loop processing is optimized.

Code generation is optimized by predicting branch conditions, preventing repetition of duplicate compare instructions.

`-O2`

Optimizes code generation by placing a higher priority on execution speed than for `-O`.

The optimization performed here includes the following processes:

Common arithmetic/logic processing in a global area is replaced by one operation. (Operations common to a global area are deleted.)

Optimization of loop processing is performed twice.

Register assignment is optimized in the operand of a simple instruction such as the `ld` instruction.

Branch condition blocks that are never reached are disregarded, and no code is generated for such blocks.

Although almost all optimization is effective, no functions are expanded in-line unless they are declared with `inline`.

`-O3`

Optimizes code generation by placing an even higher priority on execution speed than for `-O2`.

However, depending on the source code, the highest execution speed does not always result even if the `-O3` option is specified.

The optimization performed here includes the following processes:

Functions without `inline` declaration are expanded inline, and subroutines comprised of simple code are generated by copying the code of the function proper instead of calling those subroutines. This reduces the function call overhead.

`-Os`

Optimizes code generation by prioritizing size.

All optimization processes for `-O2` are performed, except those that increase code size. Code execution speed will be lower than `-O1`, `-O2`, and `-O3`.

Default: Code optimization is not 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/C++ compiler.

The debugging information is used by the debugger **gdb** only during debugging. Always specify this option.

Default: No debugging information is output.

-fPIC

Function: **Generate position-independent code**

Description: When this option is specified, the **xgcc** C/C++ compiler generates a position-independent code (PIC) suitable for dynamic linking. Specify this option when generating a library module or reentrant module.

This option cannot be specified in combination with the `-medda32` option (for C33 STD Core and C33 PE Core).

Default: No position-independent code is generated.

In addition to the standard options, the following S1C33 options are available:

-mc33adv

Function: **Generate C33 ADV Core code**

Description: When this option is specified, the **xgcc** C/C++ compiler generates the codes for C33 ADV Core models.

The C33 ADV Core instructions are generated as well as the standard instructions. Also the `%dp` register is used as the default data area pointer instead of the `%r15` register.

Default: The **xgcc** C/C++ compiler generates the codes for C33 STD Core models.

-mc33401

Function: **Assembler filter for the C33 ADV Core**

Description: The assembler instructions for the C33 ADV Core may not function normally when an interrupt is generated, depending on the combination of instructions used. This option enables an assembler filter that circumvents this problem.

Always confirm that this option is specified along with the `-mc33adv` option.

For more information on combinations of assembler instructions, etc., refer to "\gnu33\doc\as_fil_readme.txt".

Default: The assembler filter is disabled.

-mc33pe

Function: **Generate C33 PE Core code**

Description: When this option is specified, the **xgcc** C/C++ compiler generates the codes for C33 PE Core models.

The C33 PE Core instructions are generated as well as the standard instructions.

Default: The **xgcc** C/C++ compiler generates the codes for C33 STD Core models.

-medda32

Function: **Disable default data area**

Description: This option specifies an access method to external variables.

Use this option when using a 32-bit memory space.

When this option is omitted, although access range for external variables is limited to 26-bit long from the data pointer, code size is reduced.

This option is effective only for the C33 STD Core and C33 PE Core.

Table 6.3.2.1 `-medda32` option

CPU	`-medda32`	Description	C code	Assembler code	Code size
C33 STD/ C33 PE	Specified	External variables are accessed without using the default data area pointer. The compiler does not generates codes that use `%r15`. The `-fPIC` option cannot be used in combination.	`i = i_Gloval_Val;`	`xld.w %r4,i_Gloval_Val` `ld.w  %r4,[%r4]`	8 bytes

			`i_Gloval_Val = 1;`	`xld.w %r5,1 ;0x1`
				`xld.w %r4,i_Gloval_Val`
				`ld.w  [%r4],%r5`
					14 bytes

	Not specified	`%r15` is used as the default data area pointer, therefore, it must be initialized at the beginning of the program.
		Although access range for external variables is limited to 26-bit long from the data pointer, code size is reduced.
			`i = i_Gloval_Val;`	`ext   imm13-25(i_Gloval_Val)`
				`ext   imm0-12(i_Gloval_Val)`
				`ld.w  %r4,[%r15]`
					6 bytes

			`i_Gloval_Val = 1;`	`xld.w %r4,1 ;0x1`
				`ext   imm13-25(i_Gloval_Val)`
				`ext   imm0-12(i_Gloval_Val)`
				`ld.w  [%r15],%r4`
					12 bytes

C33 ADV	Invalid	This option does not affect the output code for the C33 ADV Core.
		`%dp` is used as the default data area pointer, therefore, it must be initialized at the beginning of the program.
			`i = i_Gloval_Val;`	`ext   imm19-31(i_Gloval_Val)`
				`ext   imm6-18(i_Gloval_Val)`
				`ld.w  %r4,[%dp+imm0-5(i_Gloval_Val)]`
					6 bytes

			`i_Gloval_Val = 1;`	`xld.w %r4,1 ;0x1`
				`ext   imm19-31(i_Gloval_Val)`
				`ext   imm6-18(i_Gloval_Val)`
				`ld.w  [%dp+imm0-5(i_Gloval_Val)],%r4`
					12 bytes

∗ `i`: Local variable, `i_Gloval_Val`: External variable

Default: The default data area is used.

-mlong-calls

Function: **Extend function calls**

Description: When this option is specified, the **xgcc** C/C++ compiler expands the function calls into three instructions with a signed 32-bit displacement. Use this option if the program execution area exceeds 2M bytes.

Default: Normally, all the function calls are compiled into two instructions with a signed 22-bit displacement that can branch within the range of ±2M bytes.

-mno-memcpy

Function: **Inline expansion of `strcpy` and `memcpy`**

Description: The `strcpy` and `memcpy` function calls are expanded inline.

Default: These functions are not expanded inline.

-Wall

Function: **Enables warning options**

Description: This function enables all of the following warning options.

These warning options can be individually disabled by adding "`-Wno-`". For example, to disable just the "`-Wcomment`" warning, add "`-Wno-comment`" after "`-Wall`".

`-Wchar-subscripts`

Outputs a warning when the subscript of an array is of the type "char".

`-Wcomment`

Outputs a warning when `"/*"` the starting character string for a comment line occurs inside a comment beginning with `"/*"`. Also outputs a warning when a comment starting with `"//"` ends with a backslash.

`-Wformat`

Checks whether the argument is appropriate for a converted character string when the printf or scanf function is invoked. Also checks whether the conversion specified by the converted character string is appropriate.

`-Wimplicit-int`

Outputs a warning if a format is not specified when a variable or function is declared.

`-Wimplicit-function-declaration`

Outputs a warning when a function is used before declaration.

`-Wimplicit`

Same as "`-Wimplicit-int`" and "`-Wimplicit-function-declaration`" in enabled state.

`-Wmain`

Outputs a warning when the format of the main function is incorrect. The main function has an external linkage and the return value is in int format. It should have 0, 2, or 3 arguments of the appropriate format.

`-Wmissing-braces`

Outputs a warning when parentheses are used incorrectly during initialization of arrays. For example, when a multidimensional array is initialized, a warning is output if parentheses are not used correctly for each dimension.

Example: `long l_Array_1[3][3] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };`
(Warning is output.)

`long l _ Array _ 2[3][3] = { { 0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8 } };`
(No warning is output.)

`-Wparentheses`

Outputs a warning when omission of parentheses results in ambiguities in the description. For example, a warning is output if "{ }" are omitted from a nested if statement.

`-Wsequence-point`

Outputs a warning in the case of the standard C language specification if a code described might result in undefined behavior due to the absence of an accurate execution sequence indication.

Example: `i_Array[i_Val++] = i_Val;`

`-Wreturn-type`

Outputs a warning when the return value format is defined as the default "int" format since it is not specified when the function was defined. Also outputs a warning when no value is returned when the return value is a function other than the void type.

`-Wswitch`

Outputs a warning when case statements do not exist for all enum values when the `switch` statement uses a variable of the enum type for the index. (If a default label exists, this warning is not output.) Also outputs a warning when a case statement specifies a value outside the range of enum type.

`-Wunused-function`

Outputs a warning when a static function is declared but not defined. Also outputs a warning when a static function that is not inline is defined but not used.

`-Wunused-label`

Outputs a warning when a label is declared but not used.

`-Wunused-variable`

Outputs a warning when a static variable other than local variable or const is declared but not used.

`-Wunused-value`

Outputs a warning when a calculation is performed even though the calculation result clearly will not be used.

`-Wunused`

Same as all "-Wunused-xxxx" above in the enabled state.

`-Wuninitialized`

Outputs a warning when a local variable is used without initialization. This warning is not output when `-O0` is selected.

Default: The above warning options are disabled.

`-mno-sjis-filt`

Function: **Disables the filter function for the Shift JIS code**

Description: This option disables the filter function for the Shift JIS code.
For detailed information on this filter function, refer to Section 6.8, "Filter Function for Shift JIS Code".

Default: The preprocessor performs filtering for Shift JIS code.

`-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.

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:
- The ANSI library does not use `%r15` as the default data area pointer. For the registers used by middlewares, refer to the respective middleware manual.
- 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.
- The `-medda32` and `-fPIC` options cannot be specified in combination (for C33 STD Core and C33 PE Core).

6.4 Compiler Output

This section explains the assembly sources output by the **xgcc** C/C++ compiler and the registers used by the **xgcc**.

6.4.1 Output Contents

After compiling C/C++ sources, the **xgcc** C/C++ compiler outputs the following contents:
- C33 STD Core instruction set mnemonics
- C33 ADV Core instruction set mnemonics (when the `-mc33adv` option is specified)
- C33 PE Core instruction set mnemonics (when the `-mc33pe` option is specified)
- Extended instruction mnemonics
- Assembler directives

All but the basic instructions are output using extended instructions.

Since the system control and `mac` instructions cannot be expressed in the C/C++ source, use in-line assemble by `asm` or an assembly source file to process them.
Example: `asm("mac  %r12");`

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/C++ compiler supports all data types under ANSI C/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
char	1	-128 to 127
unsigned char	1	0 to 255
short	2	-32768 to 32767
unsigned short	2	0 to 65535
int	4	-2147483648 to 2147483647
unsigned int	4	0 to 4294967295
long	4	-2147483648 to 2147483647
unsigned long	4	0 to 4294967295
pointer	4	0 to 4294967295
float	4	1.175e-38 to 3.403e+38 (normalized number)
double	8	2.225e-308 to 1.798e+308 (normalized number)
long long	8	-9223372036854775808 to 9223372036854775807
unsigned long long	8	0 to 18446744073709551615
bool	1	true/false
wchar_t (C)	2	0 to 65535
wchar_t (C++)	4	-2147483648 to 2147483647

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, a warning 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;
     →warning: integer constant is too large for "long" type
ll_val = 0x1234567812345678LL;
     →OK
```

The type `bool` is the data type needed to handle determination of `true` or `false`. C requires the inclusion of `stdbool.h`. In C++, `bool` is an independent type.

Type `wchar_t` is the data type needed to handle wide characters. In C, this data type is defined in `stdlib.h`/`stddef.h` as the type `unsigned short`.
In C++, this data type is a signed four-byte independent type. To use `wchar_t` in C++, specify the prefix `L`. This specification is required to have the compiler recognize wide character strings as such.
Example: `wchar_t buf[] = L"123";`

Store positions in memory

The positions in the memory where data is stored depend on the type. Regardless of whether it is global or local, data is located in the memory in as many bytes as are determined by the size beginning with an address that can be divided by the size.

The `double` type is aligned at 4-byte boundary addresses, so that the 4 low-order bytes of data (mantissa part (31–0)) are stored in 4 bytes of low-order locations of memory, and the 4 high-order bytes of data (sign, exponent, and mantissa part (51–32)) are stored in 4 bytes of high-order memory locations.

Structure data

Structure data is located in the memory beginning with 4-byte boundaries (addresses divided by 4) in the same way as stated above for the `double` type. 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       hData;
        char        cArray[3];
        int         iData;
        double      dData;
};
```

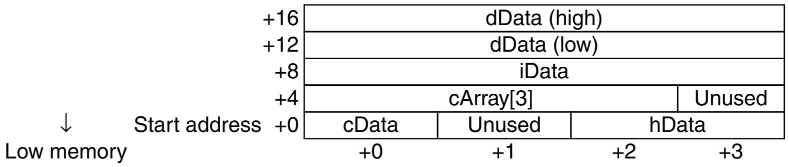


Figure 6.4.2.1 Sample locations of structure data in the memory

As shown in the diagram above, some unused areas may remain in the memory depending on the data type of a member.

6.4.3 Method of Using Registers

The following shows how the **xgcc** C/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 used as a frame pointer
	Register that need has to their values saved when calling a function
%r1	Registers that need have to their values saved when calling a function
%r2	
%r3	
%r4	Register for storing returned values (8/16/32-bit data, 32 low-order bits of double-type data)
%r5	Register for storing returned values (32 high-order bits of double-type data)
%r6	Register for passing argument (1st word)
%r7	Register for passing argument (2nd word)
%r8	Register for passing argument (3rd word)
%r9	Register for passing argument (4th word)
%r10	Scratch register/unused
%r11	
%r12	
%r13	
%r14	
%r15	Default data area pointer register*
%dp	Default data area pointer register (when the `-mc33adv` option (C33 ADV Core) is specified)

* When the `-medda32` option (default data area is not used) is not specified

Registers for saving values when calling a function (`%r0` to `%r3`)

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.

Register used as a frame pointer (`%r0`)

A frame pointer is needed to use `alloca()` or variable length arrays. In this case, this register is used as a frame pointer. However, unless a frame pointer is needed, this register is used to save a value as normally used when a function is called.

Registers for storing returned values (`%r4`, `%r5`)

These registers are used to store returned values. They are used as scratch registers before storing a returned value.

Registers for passing arguments (`%r6` to `%r9`)

These registers are used to store arguments when calling a function. Arguments exceeding the four words are stored in the stack before being passed. They are used as scratch registers before storing arguments.

Scratch registers (`%r10` to `%r14`)

These registers are used to store the temporary calculation results of expressions and local variables. These registers do not need to be saved when calling a function.

Default data area pointer register (`%r15` or `%dp`)

This register is used to store the default data area pointer value and indicates the default data area base address. The `%r15` register is used when the `-mc33adv` option is not specified. The `%dp` is used when the `-mc33adv` option is specified.
If the `-medda32` option is specified, the default data area is not used, in which case `%r15` is also not used as a default frame pointer.
However, the C33 ADV Core always uses `%dp` as a default data area, whether or not the `-medda32` option is specified.

Note: When creating assembler subroutines that are called from C/C++ routines, pay attention to the register usage.

- Before the `%r0` to `%r3` registers can be used, the contents must be saved to the stack using the `pushn` instruction. Also, the saved contents must be restored from the stack using the `popn` instruction.
- The `%r4` and `%r5` registers can be used without saving/restoring the contents until a returned value is set in the register before returning.
- The `%r6` to `%r9` registers can be used after the stored arguments are used. It is not necessary to restore the contents before returning.
- The `%r10` to `%r15` registers are reserved by the **as** assembler and **ld** linker for referencing symbols. Try to use these registers as little as possible.
- Passing arguments to the function that returns a structure data
  If the length of the structure data that is returned from a function is 8 bytes or less, the structure data is stored in the `%r4` and `%r5` registers used for storing returned values. In this case, the pointer to the structure that is normally sent as the 1st argument is not passed to the function.

6.4.4 Function Call

The way arguments are passed

When calling a function, arguments up to four words are stored in registers for passing argument (`%r6` to `%r9`) while larger arguments are stored in the stack frame of the calling function (explained in the next section) before they are passed.

Example: `func(w1,w2,w3,w4,w5,w6);`

`w1`→`%r6`, `w2`→`%r7`, `w3`→`%r8`, `w4`→`%r9`, `w5`&`w6`→Stack

(`w`*N*: arguments equal to or smaller than word size)

Basically, arguments are stored in `%r6` to `%r9` in the order that they are specified.

Data size of argument

Arguments in data size of 4 bytes or less are handled in units of words (4 bytes) irrespective of the data type. The `char` and `short` types of data are sign-extended; the `unsigned char` and `unsigned short` types are zero-extended. Only the `double` type is handled in units of 8 bytes. Unless two registers among `%r6` to `%r9` are available when passing an argument of the `double` type, it is passed via the stack.

Example: `func(w1,d2,d3,w4);`

`w1`→`%r6`, `d2`(L)→`%r7`, `d2`(H)→`%r8`, `w4`→`%r9`, `d3`→Stack

(`w`*N*: arguments equal to or smaller than word size; `d`*N*: arguments of `double` type)

Handling of structure arguments (Note)

If the argument is structure data, the values of structure members are passed via a stack.

Example:

```
struct _foo {
      int   a;
      short b;
      char  c;
};

callee(struct _foo foo,int d);
```

In the above example, only `d` is stored in the register for passing argument (`%r6`) and all the members of `foo` are stored in the stack.

Passing argument to a function that returns structure (Note)

When calling a function that returns structure data, the structure address where the result is stored is set in the `%r6` register as the first argument before being passed to the called function. Consequently, the arguments written in the source are successively carried down by one.

If the structure is not used as a returned value, the compiler assigns dummy structure data to the local variable area of the calling function and passes the address of this location.

The called function returns the pointer passed in the first argument to the calling function as a return value.

If the length of the structure data that is returned from a function is 8 bytes or less, the structure data is stored in the `%r4` and `%r5` registers used for storing returned values. In this case, the pointer to the structure that is normally sent as the 1st argument is not passed to the function.

Saving registers

If a called function modifies the `%r0` to `%r3` registers, the function has to `save` and `restore` the register values. The registers within `%r10` to `%r15` that are used as scratch registers and the `%r4` to `%r9` registers can be used without such a restriction.

Returned values

The word size or less of returned value is stored in the `%r4` register. The double-word size (`double`) of returned value is stored in the `%r4` (low-order word) and `%r5` (high-order word) registers.

Note: When a source program in which a structure is passed to or returned from a subroutine, the actual code is created so that all the members of the structure are copied using the `memcpy` function. This is undesirable since it increases the code size, lowers the execution speed and causes bugs in the compiler. Therefore, pointers should be used for passing structures as much as possible.

6.4.5 Stack Frame

When calling a function, the **xgcc** C/C++ compiler creates the stack frame shown in Figure 6.4.5.1. The start address of the stack frame is always a word (32-bit) boundary address.

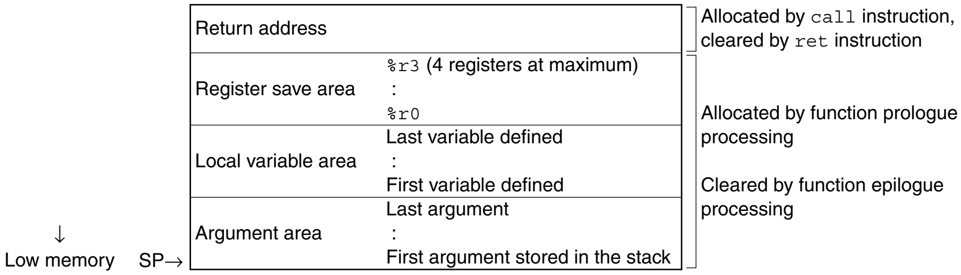

Return address
Register save area
%r3 (4 registers at maximum)
:
%r0
Local variable area
Last variable defined
:
First variable defined
Argument area
Last argument
:
First argument stored in the stack
↓
Low memory
SP→
Allocated by call instruction, cleared by ret instruction
Allocated by function prologue processing
Cleared by function epilogue processing

Figure 6.4.5.1 Stack frame

Return address

This is the return address (1 word) to the calling function.

Register save area

If any registers from `%r0` to `%r3` are used by the calling function, they are saved to this area in order of register numbers beginning with the highest.

If none of the registers from `%r0` to `%r3` is used by the calling 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 (4-byte boundaries for the `double` type) according to the data types.

Example:
```
{
        char  c;
        short s;
        int   i;
           :
}
```

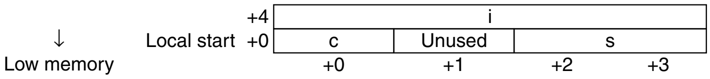

+4
i
↓
Local start +0
c
Unused
s
Low memory
+0
+1
+2
+3

Figure 6.4.5.2 Example of local variables saved to stack

This area is not allocated if there is no local variable that needs to be saved in the stack.

Argument area

If there are any arguments for another function call in the called function that cannot be stored in the registers for passing argument, an area is allocated in the stack frame (see the preceding section). All arguments are located at 4-byte boundaries. The 32 low-order `double`-type bits are saved at low-order addresses, and the 32 high-order bits are saved at the high-order addresses.

This area is not allocated if there is no function call.

Allocating and clearing the stack area

A stack area for the return address is allocated and the address is saved to this area by the `call` instruction. The address is popped from the stack and the area is cleared by the `ret` instruction.

All other areas are allocated in the prologue processing of the function, and are cleared in the epilogue processing.

6.4.6 Grammar of C/C++ Source

Refer to Section 4.2, "Grammar of C/C++ Source", for data type, library functions and header files, in-line assemble, and prototype declarations (declaring interrupt handler functions, prototype declaration with section specification).

6.5 Standard Header File and Standard Header

Files in the `\gnu33\include` directory to which `".h"` is appended are standard header files; the others are standard headers.
We recommend including a standard header file in the C source.
We recommend including a standard header in the C++ source. The standard headers support the name space `std`.

The directories of the standard header files and standard headers are comprised as described below.

```
include
     /backward
     /bits
     /c
     /c_compatibility
     /c_std
     /ext
     /machine
     /std
     /sys
```

Although multiple subdirectories are found under the `include` directory, make sure that only the `include` directory is specified for the include path. This holds for both C and C++.
Example: `-I/cygdrive/c/EPSON/gnu33/include`

Listed below are the standard headers supplied with this package. The standard headers conform to ISO/IEC 14882.

Table 6.5.1 Standard header list for C++ libraries

Header name	Description
algorithm	Defines templates for an algorithm that can be used for C++ programs.
bitset	Defines templates for a bitset container.
complex	Defines a template that handles complex numbers.
deque	Defines templates for a deque container.
exception	Defines types and functions related to exception processing.
fstream	Defines a file stream class. However, note that file streams in this package are a dummy facility.
functional	Defines templates needed to generate function objects.
iomanip	Defines an input/output manipulator.
ios	Defines an input/output stream class.
iosfwd	Declares the forward reference of an input/output stream class.
iostream	Defines a standard stream.
istream	Defines an input stream class.
iterator	Defines a replicator.
limits	Defines the numeric_limits class.
list	Defines templates for a list container.
locale	Defines a locale. However, note that locale names in this package conform only to the default "C"/"POSIX".
map	Defines a template for map and multimap containers.
memory	Defines a template that assigns objects, manages ownership, and frees objects.
new	Declares functions that allocate and deallocate memory.
numeric	Defines a template for numeric operations.
ostream	Defines an output stream class.
queue	Defines templates for queue and priority_queue containers.
set	Defines templates for set and multiset containers.
sstream	Defines an input/output string stream class.
stack	Defines a template for a stack container.
stdexcept	Defines types and functions for standard exception processing.
streambuf	Defines a stream buffer class.
strstream	Defines a string sequence input/output class.
string	Defines the basic_string class.
typeinfo	Defines a class that handles runtime type information (RTTI).
utility	Defines a comparison operator.
valarray	Defines templates for an array that handles numeric values.
vector	Defines templates for a vector container.

Table 6.5.2 Standard header list for C libraries

Header name	Description
cassert	Supports assert.h.
cctype	Supports ctype.h.
cerrno	Supports errno.h.
cfloat	Supports float.h.
ciso646	Supports iso646.h.
climits	Supports limits.h.
clocale	Supports locale.h.
cmath	Supports math.h.
csetjmp	Supports setjmp.h.
csignal	Supports signal.h.
cstdarg	Supports stdarg.h.
cstddef	Supports stddef.h.
cstdio	Supports stdio.h.
cstdlib	Supports stdlib.h.
cstring	Supports string.h.
ctime	Supports time.h.
cwchar	Supports wchar.h. However, since this package does not include wchar.h, the lines of source code corresponding to wchar.h have been changed to comments.
cwctype	Supports wctype.h. However, since this package does not include wctype.h, the lines of source code corresponding to wctype.h have been changed to comments.

Note: Not all functions declared in the above standard header list for C libraries are implemented in the ANSI library supplied with this package. For more information on implemented functions, refer to Section 7.4.2, "ANSI Library Function List."

6.6 Points to Note when Using C++

For more information on the C++ syntax, refer to the general literature on C++.
This section describes various points worth noting when using this particular implementation of C++.

Essential processing

Initialization: Clearing the `.bss` and `.comm` sections to 0.
Copying `.data` sections from ROM to RAM
Constructor of the global class
    Call the `_init()` function supplied from `libstdc++.a`.
    This must be executed immediately before the `main()` function.
Reserving storage for the heap area
    A heap area is needed for the STL (Standard Template Library) or string class to be used. Call the `init_lib()` function supplied from `lib.c`.

Termination processing: Destructor of the global class
    Call the `_fini()` function supplied from `libstdc++.a`.
    This must be executed immediately after the `main()` function.

Constructor and destructor refer to functions executed when a class is generated and degenerated. For a local class, these functions are executed when the class is generated or degenerated: i.e., at the beginning and end of the function in which the local class is declared.
In contrast, for a global class, class generation and degeneration are executed before and after the `main()` function, since this class persists in global space.

Shown below are examples of initialization and termination processing.

`_init_section();`	Clears the `.bss` and `.comm` sections to 0 Copies the `.data` sections from ROM to RAM
`_init_sys();`	Initializes simulated I/O
`init_lib();`	Reserves storage for the heap area, etc.
`_init();`	Executes the global class constructor function.
`main();`	
`_fini();`	Executes the global class destructor function.
`_exit();`	

For templates, refer to `vector.c` in the `\gnu33\utility\sample_c++` directory.
By default, the heap area is set to 0x600000 to 0x6fffff by ALLOC_START/ALLOC_END in `lib.c`. This default heap area may be corrected as needed for specific development environments.

Stream

Support is provided for the standard stream (cin/cout/cerr·clog), as is the standard input/output (stdin/stdout/stderr). This stream can be used through the simulated I/O of the debugger **gdb**.
No support is provided for the file stream handling input/output to and from files. In this package, the standard libraries for file processing such as `fopen()` and `fclose()` that are called internally in a file stream are implemented as dummy functions.

Processing performed in the debugger gdb

The debugger **gdb** supports C++. However, the following limitations apply.

- Values may be displayed in the [Watch Expressions] window for up to class member variables. However, type information cannot be acquired correctly for certain classes in the C++ library, in which case the information will be indicated as an "error type".
- When executed, the [Next] command may behave as a Step or Go command. If this presents a problem, add the `-O0` option before recompiling the program and try running it through **gdb**. This should significantly improve program behavior during [Next] execution. Then try recompiling the program with the `-O` option on release, as it is effective. Note, however, that the final program operation should always be verified after recompilation with `-O`.

Initialization of type `string`

Among the various methods available for initializing type `string`, one method involves passing pointers to variables as the initial value. This method requires caution, since if the variable used for initialization is located at address 0, the compiler may fail to initialize.

Although this package permits external variables to be located from address 0, if the external variable located from address 0 is used to initialize the type `string`, the compiler will misidentify it as a NULL pointer, causing an error by calling the `_exit()` function.

In such cases, modify memory allocation so that the external variable used for initializing the type `string` will be located at addresses other than 0.

Example: `char c_array[5];` ← Located at address 0

`std::string str1 = c_array;` ← Misidentified as a NULL pointer; calls the `_exit()` function.

6.7 Upgrading the gcc Core and Added/Removed Features

Beginning with this package (V3.0), the gcc core has been upgraded (from ver. 2.95.2 to ver. 3.3.2), with certain features added and others removed.

Added features

- Support for C++
- Support for type `long long`
- Support for `alloca()`/variable length arrays

To use `alloca()`, be sure to include `alloca.h`.
Since `alloca()` does not check for stack overflows, monitor the stack size at that point in time.
Also keep in mind that the values of `long long`-type variables and those of local variables after `alloca()`/ variable length array is used are not displayed precisely in the [Watch Expressions] window of the debugger **gdb**.

Features removed

- S/T/Z/G data areas

The S/T/Z/G data areas are no longer formally supported.
Only the default data area is normally supported. Pursuant to this change, the compiler options listed below are no longer formally supported.
`-mdp`, `-mgda`, `-mgdp`, `-mezda`, `-metda`, `-mesda`

While this feature is not extinct, it is no longer formally supported. If this feature is needed, it should be used at the user's own risk.

Differences between gcc core ver. 2.95.2 and ver. 3.3.2

There are no differences in register usage and stack frame and other basic structures between the previous and upgraded gcc core. However, the following main differences should be noted:

- If an inline assembler is declared on multiple lines, "`\n`" is required at the end of the line.

  Example:

  ver. 2.95.2)

```
asm("xxxxx
     xxxxx
     xxxxx");
```

  ver. 3.3.2)

```
asm("xxxxx\n\
     xxxxx\n\
     xxxxx");
```

- Types are checked more rigorously. If `volatile` and `const` fail to match in `extern` and body declarations, errors will result.

  Example:

```
test.h)
extern int i_Val;

test.c)
#include <test.h>
const int  i_Val;
```

  ver. 2.95.2)

```
OK
```

  ver. 3.3.2)

```
error: conflicting types for'int i_Val'
error: previous declaration as'const int i_Val'
```

- If the source file does not end with a new-line code, the following warning is generated.

```
warning: no newline at end of file
```

6.8 Filter Function for Shift JIS Code

Description of function

The original GNU preprocessor/compiler is not fully compatible with the Shift JIS code (hereafter written as the “SJIS code”). This means that for an SJIS character code like “能’ (0x945c), the “0x5c” part of the code will be incorrectly judged as a line connector (\);the character code will not be processed correctly.

Example:

`i_Val = 0;` // 機能

`i_Val = 1;` ← This line is joined to the above line and processed as a comment.

To prevent such errors, the preprocessor/compiler in package versions 3.3.0 and later incorporates a function to filter SJIS codes.

When the code “0x5c” is encountered, this filter function checks one byte immediately before that code and determines whether “0x5c” is a line connector (‘\’) or the second byte of an SJIS code.
If the function determines the code constitutes the second byte of an SJIS code, it performs a process to prevent “0x5c” from being processed as a line connector (‘\’).
This function is compatible with the following files:

- C/C++ source files
- Header files included from C/C++ files
- Assembly source files
- Header files included from assembly source files

Specify the -mno-sjis-filt option to disable this filter option.

Notes

- The -traditional-cpp option is not supported but is available in the preprocessor. The filter function will not operate properly if this option is specified while building a project.
  The -traditional-cpp option is designed to execute a preprocess in accordance with a rule in place before ISO specifications were established.
- If the filter process is enabled, the output of code for a wide character (2-byte) enclosed in single quotation marks (‘) will differ from that output by versions before 3.3.0.
  The prefix “L” must be added to ensure correct output.

Example:

Inserting a wide character (2-byte) “空” (SJIS code: 0x8bf3)

- When a version before 3.3.0 is used or when the -mno-sjis-filt option is specified in versions 3.3.0 or later

`int i_Val = ‘空’;` → i_Val is replaced by 0x8bf3

- When the -mno-sjis-filt option is not specified in versions 3.3.0 or later

`int i_Val = L’空’;` → i_Val is replaced by 0x8bf3 if the prefix “L” is specified. If the prefix “L” is not specified, i_Val is replaced by 0xfff3.

6.9 C and Assembler Mixed Programming

Control can pass between C and assembler routines as desired, providing that rules for arguments, return values, and register content protection are observed.

Creating an assembler routine called from C

```
;*****************************************************
; strcpy
;     string copy from src to dest until 0 terminate
;
; arguments : %r6:dest addr, %r7:src addr (0 terminate string)
; return    : %r4:dest addr
;*****************************************************

      .section .text
      .align   1
      .global  strcpy
      .type    strcpy,@function

strcpy:
      ld.w     %r4, %r6            ; return dest add

strcpy_loop:
      ld.ub    %r10, [%r7]+        ; copy src 1 byte to dest
      ld.b     [%r6]+, %r10
      cmp      %r10, 0             ; continue until 0 terminate
      jrne     strcpy_loop
      ret
```

This routine is called from a C routine as follows.

```
#include <string.h>
            |
int main()
    {
            |
      char *pchMem;                /* for malloc, strcpy */
            |
      strcpy(pchMem, "This is strcpy test");
    }
```

The first and the second arguments are respectively placed in the `%r6` and the `%r7` registers when passed, and the return value is stored in the `%r4` register.

As in this example, arguments and return values must be exchanged using registers, as follows:

- The first to fourth arguments are placed in the `%r6` to the `%r9` registers when passed.
- In special cases, the preceding arguments and the fifth and subsequent arguments are placed in the stack when passed. (Refer to the compiled code.)
- The return value is stored in the `%r4` register when returned.

The limitations on register usage within the assembler routine called from a C routine are as follows:

- Before the `%r0` to `%r3` registers can be used, the contents must be saved to the stack using the `pushn` instruction. Also, the saved contents must be restored from the stack using the `popn` instruction.
- The `%r4` and `%r5` registers can be used without saving/restoring the contents until a returned value is set in the register before returning.
- The `%r6` to `%r9` registers can be used after the stored arguments are used. It is not necessary to restore the contents before returning.
- The `%r10` to `%r15` registers are reserved by the **as** assembler and **ld** linker for referencing symbols. Try to use these registers as little as possible.
- Passing arguments to the function that returns a structure data
  If the length of the structure data that is returned from a function is 8 bytes or less, the structure data is stored in the `%r4` and `%r5` registers used for storing returned values. In this case, the pointer to the structure that is normally sent as the 1st argument is not passed to the function.

In the example below, the `%r0` to `%r3` registers are saved and restored before returning.

```
__adddf3:
      pushn %r3         ; save register values
           |
      popn  %r3         ; restore register values
      ret
```

Creating an assembler routine that calls a C function

C functions are compiled by the preceding rules. When creating an assembler routine that calls a C function, pay attention to the following:

Rules for delivering arguments and return values

- The first to fourth arguments are placed in the `%r6` to `%r9` registers when passed.
- The `%r4` register is used to receive the return value.

Register status at return

- The `%r0` to `%r3` registers hold the contents possessed when called.
- The `%r4` to `%r15` registers and other registers `%ahr`, `%alr`, or `%psr` may have been modified.

Sharing of header files in assembler and C

The S5U1C33001C allows header files (*.h) to be shared in the assembly and C sources. Both the assembly and C header files can be referenced by using the preprocessor directive `#include "`*filename*`.h"` before a line in which the preprocessor definition is used. When assembling assembly sources, always be sure to put them through the C/C++ compiler **xgcc** and specify the `-xassembler-with-cpp` option if the preprocessor functions are to be used. If an attempt is made to assemble them by **as** alone, an error may occur. Under no circumstances may the assembler-inherent functions be used in C sources by defining them using the `#define` directive. Nor may the C language-inherent functions be used in assembly sources by defining them using `#define`.

Bad example 1:

(header1.h)

```
#define REF_SYMBOL  .extern symbol        ← This is a function inherent in the assembler.
```

(source.c)

```
    REF_SYMBOL;                           ← A syntax error will occur.
    int main( void ) {
    }
```

Bad example 2:

(header1.h)

```
#define SUCCESS_FLAG(x)  ( x>= 0 )        ← This is a function inherent in C.
```

(source.s)

```
    cmp SUCCESS_FLAG(%r0)                 ← A syntax error will occur.
```

Do not write a preprocessor-dependent branch such as `#if`, `#ifdef`, or `#ifndef` in assembly sources. The source and the actual assembler lines may be misplaced with respect to each other's position when they are displayed in the debugger. Furthermore, when header files are to be referenced by `#include` in assembly sources, do not use `#include` in the header files to be referenced. In this case as well, the source and the actual assembler lines may be misplaced when they are displayed.

Bad example:

(header1.h)

```
#include "header2.h"
         |
```

(src.s)

```
#include "header1.h"
         |
```

Header references are nested.

Good example:

(header1.h)

```
/* #include "header2.h" */
         |
```

(src.s)

```
#include "header2.h"
#include "header1.h"
         |
```

Shown below is an example in which headers are shared.
In this example, `TASK_X`, which is defined in a common header, is used in the assembly source when registering tasks. The program here uses a macro replacement function for the constants that can be used in common in the assembly and C sources.

The header file shown below can be used in common in the assembly and C sources. The `#define` preprocessor directive is used to define only the values that will be determined when the preprocessor is executed. To define the values that will be determined by an external file or a link, the program must be written in accordance with the respective rules of C or assembler.
(shareh.h)

```
/* Macros in this header are used in both C and assembler sources. */

#define TASK_MASK               0x00ff
#define TASK_DISABLE            0x0000
#define TASK_ENABLE             0x0001
#define TASK_DEFAULT            TASK_ENABLE

#define TASK_USINGMASK          0xff00
#define TASK_N( x )             ( 0x0100 << x )
#define TASK_0                  TASK_N( 0 )
#define TASK_1                  TASK_N( 1 )
#define TASK_2                  TASK_N( 2 )
#define TASK_3                  TASK_N( 3 )
#define TASK_4                  TASK_N( 4 )
#define TASK_5                  TASK_N( 5 )
#define TASK_6                  TASK_N( 6 )
#define TASK_7                  TASK_N( 7 )
#define MAX_TASK_NUM            8

#define TASK_0_7                ( TASK_0 + TASK_1 + TASK_2 + TASK_3 \
                                  TASK_4 + TASK_5 + TASK_6 + TASK_7 )
#define TASK_NOT0               TASK_0_7 - TASK_0
#define TASK_0_3                TASK_0 * 0xf

/********************************************************************/
/* when defined, application go to start routine if exit.           */
/********************************************************************/
#define GOSTART_IFAPPEND

/********************************************************************/
/* if you change max task number, enable below line.                */
/********************************************************************/
//#define OVERRIDE_MAXTASK    8
/* new MAX_TASK_NUM ( max task number is 8. ) */
#define OVERRIDE_MAX_TASK_NUM  8

/********************************************************************/
/* if you don't use any tasks, enable below line.                   */
/********************************************************************/
//#define NO_TASK

/********************************************************************/
/* psr default setting                                              */
/********************************************************************/
/* This configuration psr is set at the top of program. */
/* if you don't permit interruption, comment here.*/
#define IE_BIT  0x10
/* set default interrupt-level( In many case, below should be 0). */
#define DEFAULT_INTERRUPT_LEVEL      0

#define DEFAULT_PSR     ( IE_BIT | ( DEFAULT_INTERRUPT_LEVEL << 8 )   )

/* dummy definition */
#define MACRO_DUMMY
#undef MACRO_DUMMY
```

In the file shown below, definitions that can be referenced only by the assembler are written. To use preprocessor definitions in the assembler, replacement by a string or numeral definition is effective.
(onlys.s)

```
#define GETTASK_H            ext doff_hi( ulTaskManage )
#define GETTASK_L            ext doff_hi( ulTaskManage )
#define GETTASK( register )  ld.w register,[%r15]
#define PUTTASK_H            ext doff_hi( ulTaskManage )
#define PUTTASK_L            ext doff_hi( ulTaskManage )
#define PUTTASK( register )  ld.w [%r15],register
```

Shown below is an example in which preprocessor definitions are used in the assembly source.

(asm.s)

```
/* Nesting header files is not supported in assembler sources. */
#include "shareh.h"
#include "onlys.h"
          |
        xld.w     %r0,DEFAULT_PSR                ← Uses a preprocessor definition
                            /* Set IE, IL. See shareh.h About DEFAULT_PSR. */

init_task:
        xld.w     %r6,TASK_0                     ← Uses a preprocessor definition
        xld.w     %r7,DO_INT
        call      add_task
        ret
          |
SOFT_INT:
        pushn     %r14

        /* delete old task0 */
        xld.w     %r6,TASK_0                     ← Uses a preprocessor definition
        call      delete_task
        /* add dummy task0 */
        xld.w     %r6,TASK_0                     ← Uses a preprocessor definition
        xld.w     %r7,vfnSetDummy
        call      add_task

        /* add exit task */
        xld.w     %r6,TASK_7                     ← Uses a preprocessor definition
        xld.w     %r7,EXIT_MAIN
        call      add_task

        popn      %r14
        reti
          |
/* exit all program */
/* --- no arguments --- */
EXIT_MAIN:
        /* read ulTaskManage */
        GETTASK_H                                ← Uses assembler-inherent
        GETTASK_L                                ← definitions
        GETTASK( %r4 )                           ←

        /* clear TASK_MASK part, and set TASK_DISABLE */
        xld.w     %r5,TASK_MASK
        not       %r5,%r5
        and       %r4,%r5

        /* write ulTaskManage */
        PUTTASK_H                                ← Uses assembler-inherent
        PUTTASK_L                                ← definitions
        PUTTASK( %r4 )                           ←
        ret
```

Shown below is an example of a header file that can only be used in C source files. The program shown here uses preprocessor definitions and a C syntax `do {...} while(1)` to create a quasi-local variable. Furthermore, an artifice is incorporated at the beginning of the file to ensure that no errors will be assumed even when the header file is referenced doubly. This approach is very effective in cases in which the mutual relationship between header files is complicated.

(csrc.h)

```
#ifndef _ONLYC_H_                                 /* _ONLYC_H_ */

#define _ONLYC_H_

#include "shareh.h"

/* macro definition changing a task bit to a task number. */
/* This definition can be used in only C-source files. */
#define TASK_NUM(  task_bit, returned_tasknum )  \
          do  { \
              unsigned long ulBuf; \
              \
              ulBuf = task_bit; \
              returned_tasknum = 0; \
              while ( (ulBuf & 0x100) == 0x0 ) { \
                  returned_tasknum++; \
                  ulBuf >>= 1; \
              }; \
          } while ( 0 );

/* #if, #ifdef or other branch prepro-instructions are
*    used in only C sources. */
#if defined( OVERRIDE_MAX_TASK_NUM )
#undef MAX_TASK_NUM
#define MAX_TASK_NUM       OVERRIDE_MAX_TASK_NUM
#endif

#ifdef NO_TASK
/* disable all tasks. */
#undef  TASK_DEFAULT
#define TASK_DEFAULT    TASK_DISABLE
#endif

#endif                                              /* _ONLYC_H_ */
```

Shown below is an example of a C source file that uses a preprocessor definition.

(csrc.c)

```
#include "onlyc.h"
          |
int main_loop( void )
    {

    while ( 1 ) {
        volatile unsigned long *ulpTask = &ulTaskManage;
        unsigned long          ulTaskBuf;
        unsigned int iTaskBit;
        int iTaskNum ;

        if ( ( *ulpTask & TASK_MASK ) == TASK_ENABLE ) {
            ulTaskBuf = *ulpTask;
            for ( iTaskNum = 0, iTaskBit = TASK_0 ;
                iTaskNum < MAX_TASK_NUM ; iTaskNum++, iTaskBit <<= 1 ) {
                if ( (ulTaskBuf & iTaskBit) != 0 ) {
                    /* Execute task if registered. */
                    if ( fnpTask[ iTaskNum ] != (void *)0x0 ) {
                        fnpTask[ iTaskNum ]();
                    }
                }
            }
        } else {
            /* illegal interruption occurred */
            break;
        }
    }

#if !defined( GOSTART_IFAPPEND )
ENDLESS_LOOP:
    goto ENDLESS_LOOP
#else
    return 0;
#endif

}

void add_task( unsigned long ulTaskFlag, void *fpFunc )
{

    int iNum = 0;

    TASK_NUM(  ulTaskFlag, iNum )

    fnpTask[ iNum ] = fpFunc;
    ulTaskManage = ulTaskManage | ulTaskFlag ;
}
          |
```

6.10 Functions of xgcc and Usage Precautions

- For details about the **xgcc** C/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.
- Refer to Chapter 4, "Source Files", how to declare interrupt handler functions.
- The `.gnu.linkonce` section and `weak` symbol are not supported.
- The `#pragma` preprocessor directive is not guaranteed to work properly. Use it at your own risk.
- For detailed information on bugs in the gcc core, refer to: "\gnu33\doc\release_history.pdf" and Section 6.11, "Known Issues".
- For information on the C99 standard supported in gcc3.3, refer to the site below.
  `http://gcc.gnu.org/gcc-3.3/c99status.html`

6.11 Known issues

The following shows the case of bugs recognized in GNU33 C/C++ Compiler.

No.1

content of bug

The following compile error occurs.

```
internal compiler error: Segmentation fault
Please submit a full bug report,
with preprocessed source if appropriate.
See <URL:http://gcc.gnu.org/bugs.html> for instructions.
```

reappearance code

```
#include <stdio.h>

int main( void )
{
        struct data *p ;

        printf( "OK\n" ) ;
        return( 0 ) ;
}

struct data {
        int i ;
};
```

workaround

1.Declare structure data type before declaration of a variable.
2.Or include <cstdio>.

compile condition

Only C++. It becomes an error when -gstabs is attached.

No.2

content of bug

The following compile error occurs.

```
In function `int main()':
error: parse error before numeric constant
```

reappearance code

```
#include <cstdio>

struct A {
        A( void ) {}
        A( int a ) {}
};
struct B {
        B( void ) {}
        B( A a ) {}
        B( A arg1, A arg2 )
        {
                printf( "OK\n" ) ;
        }
} ;

int main( void )
{
        B b( A(), A(1) ) ;
        return( 0 ) ;
}
```

workaround

```
B b( A(), A(1) );
```

is the declaration of the member function of struct B,
but it is recognized by mistake as the declaration of a function called b returning B.
The following declaration is recognized correctly.

```
B b((0,A()),A(1));
```

Refer item for 'Parse errors for "simple" code.'on http://gcc.gnu.org/bugs.html
about detailed and similar cases.

compile condition

Only C++

No.3

content of bug

The following compile error occurs.

```
internal compiler error: in pop_binding, at cp/decl.c:1428
Please submit a full bug report,
with preprocessed source if appropriate.
See <URL:http://gcc.gnu.org/bugs.html> for instructions.
```

reappearance code

```
#include <cstdio>

struct A
{
        class D {} ;
} ;
struct B
{
        typedef int D ;
} ;
struct C : public A
{
        struct E : public B {} ;
        class D {} ;
} ;

int main( void )
{
        printf( "OK\n") ;
        return( 0 ) ;
}
```

workaround

It becomes an error when the following two conditions are fulfilled.

- There are same name member classes in a derived class (struct C) and a base class (struct A).
- The member class in a derived class (struct E) is inherited, and there is a same name member variable / typedef in the base class (struct B).

It becomes the same error on http://gcc.gnu.org/ml/gcc-prs/2002-02/msg00024.html.
We recommend avoiding an unnecessary duplicate name of the class, and a member variable / typedef.

compile condition

Only C++

No.4

content of bug

At the time of comparing by if sentence, the value which reversed c is not recognized as 0.
Although else sentence(line 154)should be performed,if sentence(line 152) will be performed.

```
const signed short c = 0xffff;
const signed int c = 0xffffffff;
const signed long c = 0xffffffff;
const signed long long c = 0xffffffffffffffffLL;
```

The above brings the same result.

reappearance code

```
#include <stdio.h>

int main( void )
{
        const signed char c = 0xff;

        if( (signed int)(~c) ) {
                printf( "NG\n" );
        } else {
                printf( "OK\n” );
        }
        return( 0 ) ;
}
```

workaround

1. If const is removed, it will perform normally.
2. Or it will perform normally when you compare by if sentence after substituting for a temporary variable.
   ex)

```
        i_wk = (signed int)(~c);
        if( i_wk ){
```

3. And above showing type, it will perform normally in the case of an array.
   ex)

```
        const signed char c[2] = { 0xff,0xff };
        if( (signed int)(~c[0]) ) {
```

compile condition

Only C++

No.5

content of bug

The following compile error occurs.

```
error: invalid use of undefined type 'class A'
error: forward declaration of 'class A'
```

reappearance code

```
class A {
public:
        void proc_1(void) throw(A);
};
```

workaround

As showing above, the member function can't throw its own class by throw sentence.

compile condition

Only C++

No.6

content of bug

The following compile error occurs.

cc1.exe: out of memory allocating mmmmmmmm bytes after a total of nnnnnnnn bytes

reappearance code

```
unsigned char uc_array[] = { 0x00,0x01, ........ };

int main()
{

→ the size of array -- several hundreds thousand bytes
```

workaround

This is the error that the memory domain which the compiler has secured becomes insufficient at the time of compile.
Because the size of the array without dimension is too large.
The same error may be generated by the file with many lines of the source code.
Be small the memory domain which the compiler secures at once by dividing the array and the source code.

compile condition

C / C++

No.7

content of bug

In the following code, the value of b is not set to 126 correctly.

reappearance code

```
#include <stdio.h>

signed int a ;

int main( void )
{
        signed char tmp ;
        signed int b ;

        a = 128 ;
        tmp = a - 1 ; -- (1)
        b = tmp - 1 ; -- (2)
        if ( b != 126 ) {
                printf( "NG\n" ) ;
        } else {
                printf( "OK\n" ) ;
        }
        return( 0 ) ;
}
```

workaround

It is an error by optimization.
The processing of (1) & (2) is collected into one and compiled by optimization.
For the reason, mark extension (signed char-> signed int) of 128 (=0x80) and subtraction are carries out at once,
and 0xffffff80 - 2 are performed.
As the result, the value of b is set to -130(=0xffffff7e).
With this case, avoid an error by declaring volatile to signed char tmp;.

compile condition

C / C++
It becomes an error when the optimization level is -O or -O2 or -O3.

No.8

content of bug

The result of strcmp() does not become equal.
Kanji filter which changes a shift JIS code into ASCII escape at the time of compile is effective by the default.
When using macro of formation of a character sequence, because the character sequence is changed in the order of the following at the time of compile, it does not become an equal character sequence.

source code	str("字")	"\"字\""
	↓	
Conversion by Kanji filter	str("\x8e\x9a")	"\"\x8e\x9a\""
	↓	
Conversion by preprocessor	"\"\\x8e\\x9a\""	"\"\x8e\x9a\""

→ This bug has been resolved in Ver 3.3.0 or after.

reappearance code

```
#include <stdio.h>
#include <string.h>

#define str(a) #a // macro of formation of a character sequence

int main( void )
{
        if( !strcmp( str( "字" ), "\"字\"" ) ) {
                printf( "OK\n" ) ;
        } else {
                printf( "NG\n" ) ;
        }
        return( 0 ) ;
}
```

workaround

Invalidate Kanji filter.
When compiling on a command line, change "CC=xgcc_filt" into "CC=xgcc" in makefile(*. mak).
When compiling on IDE, invalidate the item of Kanji filter use in project property.

compile condition

C / C++
It becomes an error when Kanji filter is effective.
Kanji are Japanese characters.

THIS PAGE IS BLANK.

7 Library

This chapter explains the emulation library (floating-point number and integral remainder calculating functions), the `long long` type emulation library, the ANSI library and the C++ library included in the S1C33 Family C/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:

`libstdc++.a`	C++ library	Provides C++ features such as exception handling, runtime type information (RTTI), and STL (Standard Template Library).
`libgcc2.a`	**`long long`** type emulation library	Provides `long long`-type arithmetic/logic functions.
`libc.a`	ANSI library	Provides ANSI standard functions.
`libgcc.a`	Emulation library	Provides single-precision (32-bit) and double-precision (64-bit) floating-point arithmetic operations, comparison, type conversion, and modulo arithmetic.
`libgccP.a`	Emulation library (high-precision)	This is the high-precision version of `libgcc.a`.
`libstdio.a`	Simulated I/O library	Provides a library initialize function (see Section 7.4.3) and lower-level functions for inputs/outputs (see Section 7.4.4).

The libraries are installed in the following separate directories for each target processor.

```
\EPSON
   \gnu33
      \lib
         \std     C33 STD directory
         \pe      C33 PE directory
         \33401   S1C33401 directory
```

The above six libraries are located in the directories `std`, `pe`, and `33401`, respectively.
Select the appropriate library for the processor type.

Note: The emulation libraries (`libgcc.a` and `libgccP.a`) in this version do not support the feature to handle the S/T/Z data areas used in the earlier version. When S/T/Z areas must be used, link an emulation library located in a sub-directory in the `\gnu33\utility\interrupt_mask_emulib` directory.

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. `libstdio.a`
3. `libstdc++.a`
4. `libgcc2.a`
5. `libc.a`
6. `libgcc.a` (or `libgccP.a`)

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 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
```

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
```

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 S1C33 Family C/C++ Compiler Package contains emulation libraries (`libgcc.a` and `libgccP.a`) that supports the arithmetic operation, comparison, and type conversion of single-precision (32-bit) and double-precision (64-bit) floating-point numbers which conforms to IEEE format, and the remainder calculation of integers. The **xgcc** C/C++ compiler calls up functions from this library when a floating-point number or integral remainder calculation is performed. Since library functions exchange data via a designated general-purpose register, they can be called from an assembly source. To use emulation library functions, specify `libgcc.a` or `libgccP.a` during linkage.

The `libgccP.a` library in this package consists of the same functions as the `libgcc.a`. However, the following functions in the `libgccP.a` have higher operating accuracy than those of the `libgcc.a`.
`__adddf3`, **`__subdf3`**, **`__muldf3`**, **`__divdf3`**

The functions in the `libgcc.a` discard the digits under the effective range of the fixed-point part, while these four functions in the `libgccP.a` calculate the under part and reflect the rounded off results to the LSB of the fixed-point part. They are effective when a higher operating accuracy is required in arithmetic functions such as `sin`, `cos` and `tan`.

All source code is located in the following directory: `\gnu33\utility\lib_src\emulib`.

Registers used in the libraries

- The registers `%r0` to `%r14` are used.
- The register `%r15` is not used.
- The registers `%r0` to `%r3` are protected by saving to the stack before execution of a function and by restoring from the stack after completion of the function. In libraries using `%r10` through `%r14` as well, these registers are protected by saving to the stack.

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	`__adddf3`	Addition	`(%r5,%r4) ← (%r7,%r6) + (%r9,%r8)`
	`__subdf3`	Subtraction	`(%r5,%r4) ← (%r7,%r6) - (%r9,%r8)`
	`__muldf3`	Multiplication	`(%r5,%r4) ← (%r7,%r6) * (%r9,%r8)`
	`__divdf3`	Division	`(%r5,%r4) ← (%r7,%r6) / (%r9,%r8)`
	`__negdf2`	Sign inversion	`(%r5,%r4) ← -(%r7,%r6)`
Single-precision floating-point calculation	`__addsf3`	Addition	`%r4 ← %r6 + %r7`
	`__subsf3`	Subtraction	`%r4 ← %r6 - %r7`
	`__mulsf3`	Multiplication	`%r4 ← %r6 * %r7`
	`__divsf3`	Division	`%r4 ← %r6 / %r7`
	`__negsf2`	Sign inversion	`%r4 ← -%r6`
Type conversion	`__fixunsdfsi`	`double` → `unsigned int`	`%r4 ← (%r7,%r6)`
	`__fixdfsi`	`double` → `int`	`%r4 ← (%r7,%r6)`
	`__floatsidf`	`int` → `double`	`(%r5,%r4) ← %r6`
	`__fixunssfsi`	`float` → `unsigned int`	`%r4 ← %r6`
	`__fixsfsi`	`float` → `int`	`%r4 ← %r6`
	`__floatsisf`	`int` → `float`	`%r4 ← %r6`
	`__truncdfsf2`	`double` → `float`	`%r4 ← (%r7,%r6)`
	`__extendsfdf2`	`float` → `double`	`(%r5,%r4) ← %r6`
Floating-point comparison	`__fcmpd`	Comparison of `double` type	`%psr` change `← (%r7,%r6) - (%r9,%r8)`
	`__eqdf2`	Comparison of `double` type (a=b)	`%psr` change `← (%r7,%r6) - (%r9,%r8), %r4 ← 1 | 0 *`
	`__nedf2`	Comparison of `double` type (a≠b)	`%psr` change `← (%r7,%r6) - (%r9,%r8), %r4 ← 1 | 0 *`
	`__gtdf2`	Comparison of `double` type (a>b)	`%psr` change `← (%r7,%r6) - (%r9,%r8), %r4 ← 1 | 0 *`
	`__gedf2`	Comparison of `double` type (a≥b)	`%psr` change `← (%r7,%r6) - (%r9,%r8), %r4 ← 1 | 0 *`
	`__ltdf2`	Comparison of `double` type (a<b)	`%psr` change `← (%r7,%r6) - (%r9,%r8), %r4 ← 1 | 0 *`
	`__ledf2`	Comparison of `double` type (a≤b)	`%psr` change `← (%r7,%r6) - (%r9,%r8), %r4 ← 1 | 0 *`
	`__fcmps`	Comparison of `float` type	`%psr` change `← %r6 - %r7`
	`__eqsf2`	Comparison of `float` type(a=b)	`%psr` change `← %r6 - %r7, %r4 ← 1 | 0 *`
	`__nesf2`	Comparison of `float` type (a≠b)	`%psr` change `← %r6 - %r7, %r4 ← 1 | 0 *`
	`__gtsf2`	Comparison of `float` type (a>b)	`%psr` change `← %r6 - %r7, %r4 ← 1 | 0 *`
	`__gesf2`	Comparison of `float` type (a≥b)	`%psr` change `← %r6 - %r7, %r4 ← 1 | 0 *`
	`__ltsf2`	Comparison of `float` type (a<b)	`%psr` change `← %r6 - %r7, %r4 ← 1 | 0 *`
	`__lesf2`	Comparison of `float` type (a≤b)	`%psr` change `← %r6 - %r7, %r4 ← 1 | 0 *`

* a=(`%r7,%r6`) or `%r6`, b=(`%r9,%r8`) or `%r7`, `%r4=1` if true, `%r4=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

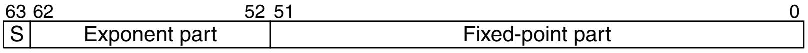
- During type conversion, values are rounded.

Floating-point format

The **xgcc** C/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)

When this type of value is stored in a register, it occupies two registers. For example, the result of a floating-point calculation is stored in the `%r5` and `%r4` registers.

`%r5` register: Sign bit, exponent part, and 20 high-order bits of fixed-point part (51:32)
`%r4` register: 32 low-order bits of fixed-point part (31:0)

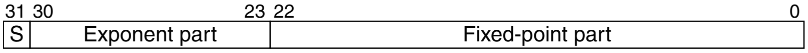
The following shows the relationship of the effective range, floating-point representation, and internal data of the `double` type.

+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)

The `float` type data can be stored in one register.
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/C++ compiler differ from the IEEE-based FPU in precision and functionality, including the manner in which infinity is handled.

7.2.3 Integral Remainder Calculation Functions

Table 7.2.3.1 below lists the integral remainder calculation functions.

Table 7.2.3.1 Integral remainder calculation functions

Classification	Function name	Functionality	
Integral division	`__divsi3`	Signed integral division	`%r4` ← `%r6 / %r7`
	`__udivsi3`	Unsigned integral division	`%r4` ← `%r6 / %r7`
Remainder calculation	`__modsi3`	Signed remainder calculation	`%r4` ← `%r6 % %r7`
	`__umodsi3`	Unsigned remainder calculation	`%r4` ← `%r6 % %r7`

These functions do not check the value of the input register. For this reason, if the `%r7` register is set to 0, a zero-division exception occurs in the `div0s` or `div0u` instruction in the function.

7.3 `long long` Type Emulation Library

7.3.1 Overview

This package contains the `long long`-type emulation library (`libgcc2.a`) to provide support for arithmetic/logic operations of type `long long`.

The C/C++ compiler, **xgcc**, calls a function in `libgcc2.a` when arithmetic/logic operations of type `long long` are performed. Since library functions exchange data via a designated general-purpose register, they can be called from an assembly source. To use `long long` type emulation library functions, specify `libgcc2.a` during linkage.

This library is based on the `libgcc2.c` supplied with the GNU C/C++ compiler ver. 3.3.2.

All source code is stored in the following directory: `\gnu33\utility\lib_src\emulib2`.

Registers used in the libraries

- The registers `%r0` to `%r14` are used.
- The register `%r15` is not used.
- The registers `%r0` to `%r3` are protected by saving to the stack before execution of a function and by restoring from the stack after completion of the function. In libraries using `%r10` through `%r14` as well, these registers are protected by saving to the stack.

7.3.2 `long long` Type Calculation Functions

Table 7.3.2.1 below lists the `long long` type calculation functions.

Table 7.3.2.1 `long long` type calculation functions

Classification	Function name	Functionality	
Calculation	`__muldi3`	Multiplication	`(%r5,%r4) ← (%r7,%r6) * (%r9,%r8)`
	`__divdi3`	Signed division	`(%r5,%r4) ← (%r7,%r6) / (%r9,%r8)`
	`__udivdi3`	Unsigned division	`(%r5,%r4) ← (%r7,%r6) / (%r9,%r8)`
	`__moddi3`	Signed remainder calculation	`(%r5,%r4) ← (%r7,%r6) % (%r9,%r8)`
	`__umoddi3`	Unsigned remainder calculation	`(%r5,%r4) ← (%r7,%r6) % (%r9,%r8)`
	`__negdi2`	Sign inversion	`(%r5,%r4) ← (%r7,%r6)`
	`__lshrdi3`	Logical shift to right	`(%r5,%r4) ← (%r7,%r6) >> (%r9,%r8)`
	`__ashrdi3`	Arithmetical shift to right	`(%r5,%r4) ← (%r7,%r6) >> (%r9,%r8)`
	`__ashldi3`	Arithmetical shift to left	`(%r5,%r4) ← (%r7,%r6) << (%r9,%r8)`
Type conversion	`__fixunsdfdi`	`double` → `unsigned long long`	`(%r5,%r4) ← (%r7,%r6)`
	`__fixdfdi`	`double` → `long long`	`(%r5,%r4) ← (%r7,%r6)`
	`__floatdidf`	`long long` → `double`	`(%r5,%r4) ← (%r7,%r6)`
	`__fixunssfdi`	`float` → `unsigned long long`	`(%r5,%r4) ← %r6`
	`__fixsfdi`	`float` → `long long`	`(%r5,%r4) ← %r6`
	`__floatdisf`	`long long` → `float`	`%r4 ← (%r7,%r6)`
Comparison	`__cmpdi2`	Comparison of `long long` type	`%psr` change ← `(%r7,%r6) - (%r9,%r8), %r4 ← 2 | 1 | 0` [*1]
	`__ucmpdi2`	Comparison of `unsigned long long` type	`%psr` change ← `(%r7,%r6) - (%r9,%r8), %r4 ← 2 | 1 | 0` [*1]
Other	`__ffsdi2`	Bit scan	`(%r5,%r4) ← (%r7,%r6)` [*2]

*1 The comparison function changes the `C`, `V`, `Z` or `N` flag of the `PSR` and `%r4` (return value) depending on the result of *op1* - *op2*, as shown below. Other flags are not changed.

Comparison result	C	V	Z	N	%r4
opt1 > opt2	0	0	0	0	2
opt1 = opt2	0	0	1	0	1
opt1 < opt2	1	0	0	1	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

7.4 ANSI Library

7.4.1 Overview

The S1C33 Family C/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 (`lib\std`, `lib\33401`, and `lib\pe`) for each C33 core type. A `long long`-type ANSI library is included in `libgcc2.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` `assert.h` `locale.h`

The following C++ headers corresponding to the above header files are installed in the `\include` directory. The C++ headers supports the namespace `std`.

`cstdio` `cstdlib` `ctime` `cmath` `cerrno` `cflaot` `climits` `cctype`
`cstring` `cstdarg` `csetjmp` `cstddef` `cassert` `clocale`

Refer to the sample file located in the `\gnu33\utility\sample_std\ansilib` directory for how to use the library functions. All source code is stored in the following directories: `\gnu33\utility\lib_src\ansilib(libc.a)` or `\gnu33\utility\lib_src\emulib2(libgcc2.a)`. Refer to the source for more information and modify them if needed.

Registers used in the library

- The registers `%r0` to `%r11` are used.
- The registers `%r12` to `%r15` are not used.
- The registers `%r0` to `%r3` are protected by saving to the stack before execution of a function and by restoring from the stack after completion of the function.

7.4.2 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 7.4.2.1 Input/output functions

Header file: **`stdio.h/cstdio`**

Function	Functionality	Reentrant	Notes
`FILE *`**`fopen`**`(const char *filename, const char *mode);`	Dummy	Dummy	
`FILE *`**`freopen`**`(const char *filename, const char *mode, FILE *stream);`	Dummy	Dummy	
`int `**`fclose`**`(FILE *stream);`	Dummy	Dummy	
`int `**`fflush`**`(FILE *stream);`	Dummy	Dummy	
`int `**`fseek`**`(FILE *stream, long offset, int origin);`	Dummy	Dummy	
`long `**`ftell`**`(FILE *stream);`	Dummy	Dummy	
`void `**`rewind`**`(FILE *stream);`	Dummy	Dummy	
`int `**`fgetpos`**`(FILE *stream, fpos_t *ptr);`	Dummy	Dummy	
`int `**`fsetpos`**`(FILE *stream, const fpost_t *pos);`	Dummy	Dummy	
`size_t `**`fread`**`(void *ptr, size_t size, size_t count, FILE *stream);`	Input array element from stdin.	Conditional	Refer to global variables `stdin` and `_iob`, and call `read` function.

Function	Functionality	Reentrant	Notes
size_t **fwrite**(const void *ptr, size_t size, size_t count, FILE *stream);	Output array element to stdout.	Conditional	Refer to global variables `stdout`, `stderr` and `_iob`, and call `write` function.
int **fgetc**(FILE *stream);	Input one character from stdin.	Conditional	Refer to global variables `stdin` and `_iob`, and call `read` function.
int **getc**(FILE *stream);	Input one character from stdin.	Conditional	Refer to global variables `stdin` and `_iob`, and call `read` function.
int **getchar**(void);	Input one character from stdin.	Conditional	Refer to global variables `stdin` and `_iob`, and call `read` function.
int **ungetc**(int c, FILE *stream);	Push one character back to input buffer.	Nonreentrant	Refer to global variables `stdin`, `stdout`, `stderr`, and `_iob`, returned value overwrite.
char ***fgets**(char *s, int n, FILE *stream);	Input character string from stdin.	Conditional	Refer to global variables `stdin` and `_iob`, and call `read` function.
char ***gets**(char *s);	Input character string from stdin.	Conditional	Refer to global variables `stdin` and `_iob`, and call `read` function.
int **fputc**(int c, FILE *stream);	Output one character to stdout.	Conditional	Refer to global variables `stdout`, `stderr` and `_iob`, and call `write` function.
int **putc**(int c, FILE *stream);	Output one character to stdout.	Conditional	Refer to global variables `stdout`, `stderr` and `_iob`, and call `write` function.
int **putchar**(int c);	Output one character to stdout.	Conditional	Refer to global variables `stdout`, `stderr` and `_iob`, and call `write` function.
int **fputs**(char *s, FILE *stream);	Output character string to stdout.	Conditional	Refer to global variables `stdout`, `stderr` and `_iob`, and call `write` function.
int **puts**(char *s);	Output character string to stdout.	Conditional	Refer to global variables `stdout`, `stderr` and `_iob`, and call `write` function.
int **remove**(const char *filename);	Dummy	Dummy	
int **rename**(const char *oldname, const char *newname);	Dummy	Dummy	
void **setbuf**(FILE *stream, char *buf);	Dummy	Dummy	
int **setvbuf**(FILE *stream, char *buf, int type, size_t size);	Dummy	Dummy	
FILE ***tmpfile**(void);	Dummy	Dummy	
char ***tmpnam**(char *buf);	Dummy	Dummy	
int **feof**(FILE *stream);	Dummy	Dummy	
int **ferror**(FILE *stream);	Dummy	Dummy	
void **clearerr**(FILE *stream);	Dummy	Dummy	
void **perror**(const char *s);	Output error information to stdout.	Nonreentrant	Refer to global variables `stdout` and `_iob`, change `errno`, and call `read` function.
int **fscanf**(FILE *stream, const char *format, ...);	Input from stdin with format specified.	Nonreentrant	Refer to global variables `stdout` and `_iob`, change `errno`, and call `read` function.
int **scanf**(const char *format, ...);	Input from stdin with format specified.	Nonreentrant	Refer to global variables `stdout` and `_iob`, change `errno`, and call `read` function.
int **sscanf**(const char *s, const char *format, ...);	Input from character string with format specified.	Nonreentrant	Change global variable `errno`.
int **fprintf**(FILE *stream, const char *format, ...);	Output to stdout with format specified.	Conditional	Refer to global variables `stdout`, `stderr` and `_iob`, and call `write` function.
int **printf**(const char *format, ...);	Output to stdout with format specified.	Conditional	Refer to global variables `stdout`, `stderr` and `_iob`, and call `write` function.
int **sprintf**(char *s, const char *format, ...);	Output to array with format specified.	Reentrant	
int **vfprintf**(FILE *stream, const char *format, va_list arg);	Output conversion result to stdout.	Conditional	Refer to global variables `stdout`, `stderr` and `_iob`, and call `write` function.
int **vprintf**(const char *format, va_list arg);	Output conversion result to stdout.	Conditional	Refer to global variables `stdout`, `stderr` and `_iob`, and call `write` function.
int **vsprintf**(char *s, const char *format, va_list arg);	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 7.4.4 for more information.

Utility functions

The table below lists the utility functions included in `libc.a` and `libgcc2.a`.

Table 7.4.2.2 Utility functions (`libc.a`)

Header file: **`stdlib.h/cstdlib`**

Function	Functionality	Reentrant	Notes
`void *`**`malloc`**`(size_t size);`	Allocate area.	Nonreentrant	Change global variables `errno`, `ansi_ucStartAlloc`, `ansi_ucEndAlloc`, `ansi_ucNxtAlcP`, `ansi_ucTblPtr` and `ansi_ulRow`.
`void *`**`calloc`**`(size_t elt_count, size_t elt_size);`	Allocate array area.	Nonreentrant	Invalid for call from memory allocate.
`void `**`free`**`(void *ptr);`	Clear area.	Nonreentrant	Invalid for call from memory allocate.
`void *`**`realloc`**`(void *ptr, size_t size);`	Change area size.	Nonreentrant	Invalid for call from memory allocate.
`int `**`system`**`(const char *command);`	Dummy	Dummy	
`void `**`exit`**`(int status);`	Terminate program normally.	Reentrant	Refer to `exit`, terminates on the side of called later.
`void `**`abort`**`(void);`	Terminate program abnormally.	Reentrant	Refer to `exit`, terminates on the side of called later.
`int `**`atexit`**`(void (*func)(void));`	Dummy	Dummy	
`char *`**`getenv`**`(const char *str);`	Dummy	Dummy	
`void *`**`bsearch`**`(const void *key, const void *base, size_t count, size_t size, int (*compare)(const void *, const void *));`	Binary search.	Reentrant	
`void `**`qsort`**`(void *base, size_t count, size_t size, int (*compare)(const void *, const void *));`	Quick sort.	Reentrant	
`int `**`abs`**`(int x);`	Return absolute value (`int` type).	Reentrant	
`long `**`labs`**`(long x);`	Return absolute value (`long` type).	Reentrant	
`div_t `**`div`**`(int n, int d);`	Divide `int` type.	Nonreentrant	Change global variables `errno`.
`ldiv_t `**`ldiv`**`(long n, long d);`	Divide `long` type.	Nonreentrant	Change global variables `errno`.
`int `**`rand`**`(void);`	Return pseudo-random number.	Nonreentrant	Change global variables `errno`.
`void `**`srand`**`(unsigned int seed);`	Set seed of pseudo-random number.	Nonreentrant	Change global variables `errno`.
`long `**`atol`**`(const char *str);`	Convert character string into `long` type.	Nonreentrant	Change global variables `errno`.
`int `**`atoi`**`(const char *str);`	Convert character string into `int` type.	Nonreentrant	Change global variables `errno`.
`double `**`atof`**`(const char *str);`	Convert character string into `double` type.	Nonreentrant	Change global variables `errno`.
`double `**`strtod`**`(const char *str, char **ptr);`	Convert character string into `double` type.	Nonreentrant	Change global variables `errno`.
`long `**`strtol`**`(const char *str, char **ptr, int base);`	Convert character string into `long` type.	Nonreentrant	Change global variables `errno`.
`unsigned long `**`strtoul`**`(const char *str, char **ptr, int base);`	Convert character string into `unsigned long` type.	Nonreentrant	Change global variables `errno`.

Table 7.4.2.3 Utility functions (`libgcc2.a`)

Header file: **`stdlib.h/cstdlib`**

Function	Functionality	Reentrant	Notes
`long long `**`strtoll`**`(const char *str, char **endptr, int base);`	Convert character string into `long long` type.	Nonreentrant	Change global variables `errno`.
`unsigned long long `**`strtoull`**`(const char *str, char **endptr, int base);`	Convert character string into `unsigned long long` type.	Nonreentrant	Change global variables `errno`.

Non-local branch functions

The table below lists the non-local branch functions included in `libc.a`.

Table 7.4.2.4 Non-local branch functions

Header file: **`setjmp.h/csetjmp`**

Function	Functionality	Reentrant	Notes
`int `**`setjmp`**`(jmp_buf env);`	non-local branch	Reentrant	
`void `**`longjmp`**`(jmp_buf env, int status);`	non-local branch	Reentrant	

Date and time functions

The table below lists the date and time functions included in `libc.a`.

Table 7.4.2.5 Date and time functions

Header file: **`time.h/ctime`**

Function	Functionality	Reentrant	Notes
`clock_t` **`clock`**`(void);`	Dummy	Dummy	
`char *`**`asctime`**`(const struct tm *ts);`	Dummy	Dummy	
`char *`**`ctime`**`(const time_t *timeptr);`	Dummy	Dummy	
`double` **`difftime`**`(time_t ti, time_t t2);`	Dummy	Dummy	
`struct tm *`**`gmtime`**`(const time_t *t);`	Convert calendar time to standard time.	Nonreentrant	Change static variable.
`struct tm *`**`localtime`**`(const time_t *t);`	Dummy	Dummy	
`time_t` **`mktime`**`(struct tm *tmptr);`	Convert standard time to calendar time.	Nonreentrant	Change static variable. The locale information and summer time setting cannot take effect.
`time_t` **`time`**`(time_t *tptr);`	Return current calendar time.	Conditional	Refer to global variable `gm_sec`.

Mathematical functions

The table below lists the mathematical functions included in `libc.a`.

Table 7.4.2.6 Mathematical functions

Header files: **`math.h`**, **`errno.h`**, **`float.h`**, **`limits.h/cmath`**, **`cerrno`**, **`cfloat`**, **`climits`**

Function	Functionality	Reentrant	Notes
`double` **`fabs`**`(double x);`	Return absolute value (`double` type).	Reentrant	
`double` **`ceil`**`(double x);`	Round up `double`-type decimal part.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`floor`**`(double x);`	Round down `double`-type decimal part.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`fmod`**`(double x, double y);`	Calculate `double`-type remainder.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`exp`**`(double x);`	Exponentiate (e^x).	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`log`**`(double x);`	Calculate natural logarithm.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`log10`**`(double x);`	Calculate common logarithm.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`frexp`**`(double x, int *nptr);`	Return mantissa and exponent of floating-point number.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`ldexp`**`(double x, int n);`	Return floating-point number from mantissa and exponent.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`modf`**`(double x, double *nptr);`	Return integer and decimal parts of floating-point number.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`pow`**`(double x, double y);`	Calculate x^y.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`sqrt`**`(double x);`	Calculate square root.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`sin`**`(double x);`	Calculate sine.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`cos`**`(double x);`	Calculate cosine.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`tan`**`(double x);`	Calculate tangent.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`asin`**`(double x);`	Calculate arcsine.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`acos`**`(double x);`	Calculate arccosine.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`atan`**`(double x);`	Calculate arctangent.	Nonreentrant	Destruct `ALR` and `AHR`.
`double` **`atan2`**`(double y, double x);`	Calculate arctangent of y/x.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`sinh`**`(double x);`	Calculate hyperbolic sine.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`cosh`**`(double x);`	Calculate hyperbolic cosine.	Nonreentrant	Change global variable `errno`, destruct `ALR` and `AHR`.
`double` **`tanh`**`(double x);`	Calculate hyperbolic tangent.	Nonreentrant	Destruct `ALR` and `AHR`.

Character functions

The table below lists the character functions included in `libc.a`.

Table 7.4.2.7 Character functions

Header file: **`string.h/cstring`**

Function	Functionality	Reentrant	Notes
`void *memchr(const void *s, int c, size_t n);`	Return specified character position in the storage area.	Reentrant	
`int memcmp(const void *s1, const void *s2, size_t n);`	Compare storage areas.	Reentrant	
`void *memcpy(void *s1, const void *s2, size_t n);`	Copy the storage area.	Reentrant	
`void *memmove(void *s1, const void *s2, size_t n);`	Copy the storage area (overlapping allowed).	Reentrant	
`void *memset(void *s, int c, size_t n);`	Set character in the storage area.	Reentrant	
`char *strcat(char *s1, const char *s2);`	Concatenate character strings.	Reentrant	
`char *strchr(const char *s, int c);`	Return specified character position found first in the character string.	Reentrant	
`int strcmp(const char *s1, const char *s2);`	Compare character strings.	Reentrant	
`char *strcpy(char *s1, const char *s2);`	Copy character string.	Reentrant	
`size_t strcspn(const char *s1, const char *s2);`	Return number of characters from the beginning of the character string until the specified character appears (multiple choices).	Reentrant	
`char *strerror(int code);`	Return error message character string.	Reentrant	
`size_t strlen(const char *s);`	Return length of character string.	Reentrant	
`char *strncat(char *s1, const char *s2, size_t n);`	Concatenate character strings (number of characters specified).	Reentrant	
`int strncmp(const char *s1, const char *s2, size_t n);`	Compare character strings (number of characters specified).	Reentrant	
`char *strncpy(char *s1, const char *s2, size_t n);`	Copy the character string (number of characters specified).	Reentrant	
`char *strpbrk(const char *s1, const char *s2);`	Return specified character position (multiple choices) found first in the character string.	Reentrant	
`char *strrchr(const char *str, int c);`	Return specified character position found last in the character string.	Reentrant	
`size_t strspn(const char *s1, const char *s2);`	Return number of characters from the beginning of the character string until the non-specified character appears (multiple choices).	Reentrant	
`char *strstr(const char *s1, const char *s2);`	Return position where the specified character string appeared first.	Reentrant	
`char *strtok(char *s1, const char *s2);`	Divide the character string into tokens.	Nonreentrant	Change static variable.

∗ All functions except `strerror` have been created and tuned by an assembly source.

Character type determination/conversion functions

The table below lists the character type determination/conversion functions included in `libc.a`.

Table 7.4.2.8 Character type determination/conversion functions

Header file: **`ctype.h/cctype`**

Function	Functionality	Reentrant
`int isalnum(int c);`	Determine character type (decimal or alphabet).	Reentrant
`int isalpha(int c);`	Determine character type (alphabet).	Reentrant
`int iscntrl(int c);`	Determine character type (control character).	Reentrant
`int isdigit(int c);`	Determine character type (decimal).	Reentrant
`int isgraph(int c);`	Determine character type (graphic character).	Reentrant
`int islower(int c);`	Determine character type (lowercase alphabet).	Reentrant
`int isprint(int c);`	Determine character type (printable character).	Reentrant
`int ispunct(int c);`	Determine character type (delimiter).	Reentrant
`int isspace(int c);`	Determine character type (null character).	Reentrant
`int isupper(int c);`	Determine character type (uppercase alphabet).	Reentrant
`int isxdigit(int c);`	Determine character type (hexadecimal).	Reentrant
`int tolower(int c);`	Convert character type (uppercase alphabet → lowercase).	Reentrant
`int toupper(int c);`	Convert character type (lowercase alphabet → uppercase).	Reentrant

Variable argument macros

The table below lists the variable argument macros defined in `stdarg.h`.

Table 7.4.2.9 Variable argument macros

Header file: **`stdarg.h/cstdarg`**

Macro	Functionality
`void` **`va_start`**`(va_list ap, type lastarg);`	Initialize the variable argument group.
`type` **`va_arg`**`(va_list ap, type);`	Return the actual argument.
`void` **`va_end`**`(va_list ap);`	Return normally from the variable argument function.

Test macro

The table below lists the test macro included in `libc.a`.

Table 7.4.2.10 Test macro

Header file: **`assert.h/cassert`**

Macro	Functionality	Reentrant	Notes
`void` **`assert`**`(int test);`	If the condition expression is false, the compiler outputs a diagnostic message before aborting the process.	Conditional	Call `printf`/ `abort` function.

Locale function

The table below lists the locale function included in `libc.a`.

Table 7.4.2.11 Locale function

Header file: **`locale.h/locale`**

Function	Functionality	Reentrant	Notes
`char *`**`setlocale`**`(int category, const char *locale);`	Dummy	Dummy	
`struct lconv *`**`localeconv`**`(void);`	Dummy	Dummy	

7.4.3 Declaring and Initializing Global Variables

The ANSI library functions reference the global variables listed in Table 7.4.3.1. These variables have been defined in the `libstdio.a` library. Include "`\include\libstdio.h`" in the C/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 "`\gnu33\sample_ide\pe\simulator\simulatedIO`".

Table 7.4.3.1 Global variables required of declaration

Global variable	Initial setting	Related header file/function

`FILE _iob[FOPEN_MAX +1];`
`FOPEN_MAX=3`, defined in `stdio.h`
File structure data for standard input/output streams
	`_iob[`*N*`]._flg = _UGETN;`
	`_iob[`*N*`]._buf = 0;`
	`_iob[`*N*`]._fd = `*N*`;`
	(*N*=0–2)
	`_iob[0]`: Input data for `stdin`
	`_iob[1]`: Output data for `stdout`
	`_iob[2]`: Output data for `stderr`
		`stdio.h`, **`smcvals.h`**
		`fgets`, `fread`, `fscanf`, `getc`, `getchar`, `gets`, `scanf`, `ungetc`, `perror`, `fprintf`, `fputs`, `fwrite`, `printf`, `putc`, `putchar`, `puts`, `vfprintf`, `vprintf`

`FILE *stdin;` Pointer to standard input/output file structure data `_iob[0]`	`stdin = &_iob[0];`	**`stdio.h`** `fgets`, `fread`, `fscanf`, `getc`, `getchar`, `gets`, `scanf`, `ungetc`
`FILE *stdout;` Pointer to standard input/output file structure data `_iob[1]`	`stdout = &_iob[1];`	**`stdio.h`** `fprintf`, `fputs`, `fwrite`, `printf`, `putc`, `putchar`, `puts`, `vfprintf`, `vprintf`
`FILE *stderr;` Pointer to standard input/output file structure data `_iob[2]`	`stderr = &_iob[2];`	**`stdio.h`** `fprintf`, `fputs`, `fwrite`, `printf`, `perror`, `putc`, `putchar`, `puts`, `vfprintf`, `vprintf`

`int errno;`
Variable to store error number
	`errno = 0;`	**`errno.h`**
		`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`

`unsigned int seed;` Variable to store seed of random number	`seed = 1;`	**`stdlib.h`** `rand`, `srand`
`time_t gm_sec;` Elapsed time of timer function in seconds from 0:00:00 on January 1, 1970	`gm_sec = -1;`	**`time.h`** `time`

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.
Before actually using this function, refer to `\gnu33\sample_ide\pe\simulator\simulatedIO\vector.c`.

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.)

7.4.4 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/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 "`\gnu33\sample_ide\pe\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 `putcc`)
`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`

7.5 C++ Library

7.5.1 Overview

The S1C33 Family C/C++ Compiler Package contains an C++ library (`libstdc++.a`).
This library provides the following facilities.

- Global class constructor and destructor, `_init()`/`_fini()`
- Exception handling
- Runtime type information (RTTI)
- Type `string`
- Memory allocation and deallocation, `new`/`delete`
- Stream
  File streams are implemented as dummy facilities.
  For the standard stream to be used, the `read` and the `write` functions are required for `cin` and `cout`/`cerr`/`clog`, respectively.
- Locale
  Locale names are supported only for default "C"/"POSIX" locales.
  Locale name changes are not supported.
- STL (Standard Template Library)
  The precise scope of the STL is not formally defined.
  In this manual, the following items are classified as STL.
  - Container (`deque`/`list`/`map`/`multimap`/`multiset`/`priority_queue`/`queue`/`set`/`stack`/`vector`)
  - Algorithm
  - Replicator
  - Allocator
  - Function object
- Other classes, `bitset`/`complex`/`valarray`

In the C++ library provided with this package, the following facilities do not support type `wchar_t`.

- Type `string`
  `wstring`/`char_traits` class
- Stream
  `wcin`/`wcerr`/`wclog`
  `wios`/`wstreambuf`/`wistream`/`wiostream`/`wstringbuf`/`wistringstream`/
  `wostringstream`/`wstringstream`/`wfilebuf`/`wofstream`/`wfstream`/`wstreampos`
- Locale
  `moneypunct` class/`moneypunct_byname` class/`money_get` class/`money_put` class/
  `numpunct` class/`numpunct_byname` class/`num_get` class/`num_put` class/`__timepunct` class/
  `time_put` class/`time_put_byname` class/`time_get` class/`time_get_byname` class/
  `messages` class/`messages_byname` class/`ctype` class/`codecvt` class/`ctype_byname` class/
  `codecvt_byname` class/`collate` class/`collate_byname` class

For information on the headers required for each facility, refer to the standard header list for the C++ library in Table 7.5.1.1 below.
For more information on using the C++/STL facilities, refer to the general literature on C++/STL.
An example of use of the C++ library is provided for reference purposes in the `\gnu33\utility\sample_c++` directory.

Table 7.5.1.1 Standard header list for C++ libraries

Header name	Description
algorithm	Defines templates for an algorithm that can be used for C++ programs.
bitset	Defines templates for a bitset container.
complex	Defines a template that handles complex numbers.
deque	Defines templates for a deque container.
exception	Defines types and functions related to exception processing.
fstream	Defines a file stream class. However, note that file streams in this package are a dummy facility.
functional	Defines templates needed to generate function objects.
iomanip	Defines an input/output manipulator.
ios	Defines an input/output stream class.
iosfwd	Declares the forward reference of an input/output stream class.
iostream	Defines a standard stream.
istream	Defines an input stream class.
iterator	Defines a replicator.
limits	Defines the numeric_limits class.
list	Defines templates for a list container.
locale	Defines a locale. However, note that locale names in this package conform only to the default "C"/"POSIX".
map	Defines a template for map and multimap containers.
memory	Defines a template that assigns objects, manages ownership, and frees objects.
new	Declares functions that allocate and deallocate memory.
numeric	Defines a template for numeric operations.
ostream	Defines an output stream class.
queue	Defines templates for queue and priority_queue containers.
set	Defines templates for set and multiset containers.
sstream	Defines an input/output string stream class.
stack	Defines a template for a stack container.
stdexcept	Defines types and functions for standard exception processing.
streambuf	Defines a stream buffer class.
strstream	Defines a string sequence input/output class.
string	Defines the basic_string class.
typeinfo	Defines a class that handles runtime type information (RTTI).
utility	Defines a comparison operator.
valarray	Defines templates for an array that handles numeric values.
vector	Defines templates for a vector container.

This library is primarily based on the source code in the `libstdc++-v3` directory provided with GNU C/C++ compiler ver. 3.3.2.
All source codes are located in the `\gnu33\utility\lib_src\libstdc++\src` directory.

7.5.2 List of STL Container Member Functions

Listed below are the member functions of the STL containers supported in the C++ library in this package.

Table 7.5.2.1 List of container member functions

		Container name									
		`deque`	`list`	`map`	`multimap`	`multiset`	`priority_queue`	`queue`	`set`	`stack`	`vector`
Member function	`assign`	○	○								○
	`at`	○									○
	`back`	○	○					○			○
	`begin`	○	○	○	○	○			○		○
	`capacity`										○
	`clear`	○	○	○	○	○			○		○
	`count`			○	○	○			○		
	`empty`	○	○	○	○	○	○	○	○	○	○
	`end`	○	○	○	○	○			○		○
	`equal_range`			○	○	○			○		
	`erase`	○	○	○	○	○			○		○
	`find`			○	○	○			○		
	`front`	○	○					○			○
	`get_allocator`	○	○	○	○	○			○		○
	`insert`	○	○	○	○	○			○		○
	`key_comp`			○	○	○			○		
	`lower_bound`			○	○	○			○		
	`max_size`	○	○	○	○	○			○		○
	`merge`		○								
	`operator[]`	○		○							○
	`pop`						○	○		○	
	`pop_back`	○	○								○
	`pop_front`	○	○								
	`push`						○	○		○	
	`push_back`	○	○								○
	`push_front`	○	○								
	`rbegin`	○	○	○	○	○			○		○
	`remove`		○								
	`remove_if`		○								
	`rend`	○	○	○	○	○			○		○
	`reserve`										○
	`resize`	○	○								○
	`reverse`		○								
	`size`	○	○	○	○	○	○	○	○	○	○
	`sort`		○								
	`splice`		○								
	`swap`	○	○	○	○	○			○		○
	`top`						○			○	
	`unique`		○								
	`upper_bound`			○	○	○			○		
	`value_comp`			○	○	○			○		

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/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

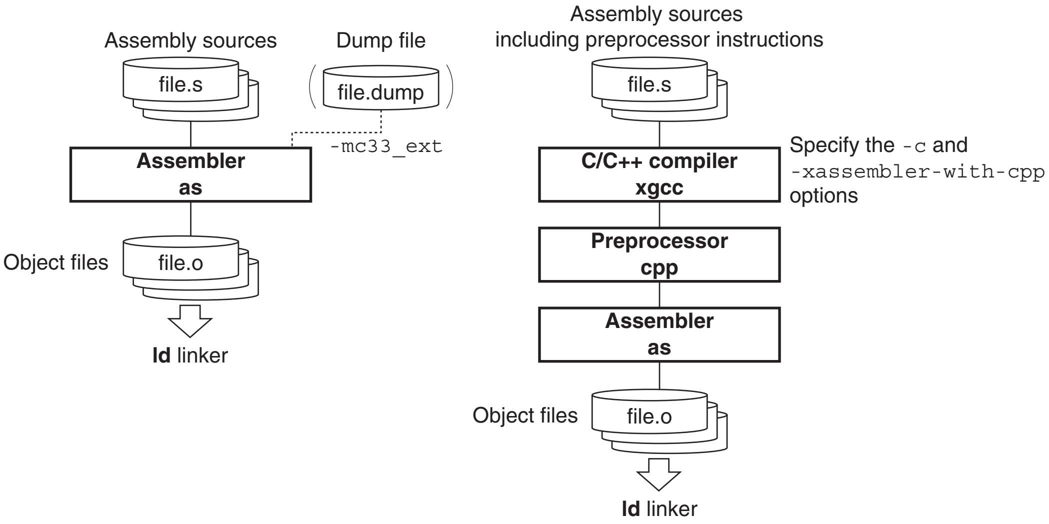

Assembly sources
Dump file
file.s
file.dump
-mc33_ext
Assembler
as
Object files
file.o
ld linker
Assembly sources
including preprocessor instructions
file.s
C/C++ compiler
xgcc
Specify the -c and
-xassembler-with-cpp
options
Preprocessor
cpp
Assembler
as
Object files
file.o
ld linker

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/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.

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 symbols when the `-mc33_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.
Input *<file name>* 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: `-adhl` 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 S1C33 option is available:

`-mc33adv`

Function: **Generate C33 ADV Core code**

Description: When this option is specified, the **as** assembler generates C33 ADV Core instructions as well as standard instructions.

Default: C33 STD Core instructions only are generated.

`-mc33401`

Function: **Assembler filter for the C33 ADV Core**

Description: The assembler instructions for the C33 ADV Core may not function normally when an interrupt is generated, depending on the combination of instructions used. This option enables an assembler filter that circumvents this problem.
Always confirm that this option is specified along with the `-mc33adv` option.
For more information on combinations of assembler instructions, etc., refer to "\gnu33\doc\as_fil_readme.txt".

Default: The assembler filter is disabled.

`-mc33pe`

Function: **Generate C33 PE Core code**

Description: When this option is specified, the **as** assembler generates C33 PE Core instructions as well as standard instructions.

Default: C33 STD Core instructions only are generated.

`-medda32`

Function:	**Disable default data area**
Description:	The data access area size is made 32 bits without using the default data area. This allows data to be located anywhere in the 4G space without regard for the data pointer. This specification eliminates the need to initialize the default data pointer, as long as the core is C33 STD or C33 PE. (Necessary for the C33 ADV Core)
Default:	The default data area is used.

`-mc33_ext` ***<dump file name> <all-object dump 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 8.7 and 8.8.
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 -mc33_ext test.dump allobjects.dump -o test.o 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

```
        ext      SYMBOL@rh
        ext      SYMBOL@m
        call     SYMBOL@rl      ...Symbol externally referred
             :
        ext      VAR1@h
        ext      VAR1@m
        ld.w     %r1,VAR1@l     ...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,"a"
.section .data,"aw"
```

Description

(1) `.section .rodata,"a"`

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. `"a"` is a flag for the `.section` directive and it means that the section is allocatable. This section is defined for read-only data. Usually, this section will be mapped into a read-only memory at the stage of linkage.

Example: `.section .rodata,"a"` Defines a `.rodata` section.

(2) `.section .data,"aw"`

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. `"aw"` is a flag for the `.section` directive and it means that the section is allocatable and writable. This section allows the program to rewrite the contents. 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,"aw"` 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`
4 bytes: `.int`, `.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 Comm Section Defining Directive (`.comm`)

List of comm section defining directives

`.comm`	Declares a symbol assigned to the `.comm` section.

Instruction format

`.comm` ***<symbol>*****`,`*****<size>*****`,`*****<alignment>***

Description

Declares symbols assigned to the `.comm` section.

However, only symbols not declared as `.local` at the immediately preceding position can be assigned to the `.comm` section. Symbols declared as `.local` at the immediately preceding position are assigned to the `.bss` section.

This section may be generated when the C/C++ compiler compiles C++ sources. Do not use this directive in the user program.

8.5.5 Ctors Section Defining Directive (`.ctors`)

List of ctors section defining directives

`.ctors`	Declares the `.ctors` section in which a pointer array to the global class constructor function is located.

Instruction format

```
.section .ctors,"aw"
```

Description

Declares the start of a `.ctors` section. Statements following this instruction are assembled as those to be mapped in the `.ctors` section, until another section is declared.

Note that "`aw`" is the flag for the `.section` directive command that specifies data locations and enables write operations, for which a section with read/write attributes is created.

This section may be generated when the C/C++ compiler compiles C++ sources. Do not use this directive in the user program.

8.5.6 Dtors Section Defining Directive (`.dtors`)

List of dtors section defining directives

`.dtors`	Declares the `.dtors` section in which a pointer array to the global class destructor function is located.

Instruction format

```
.section .dtors,"aw"
```

Description

Declares the start of a `.dtors` section. Statements following this instruction are assembled as those to be mapped in the `.dtors` section, until another section is declared.

Note that "`aw`" is the flag for the `.section` directive command that specifies data locations and enables write operations, for which a section with read/write attributes is created.

This section may be generated when the C/C++ compiler compiles C++ sources. Do not use this directive in the user program.

8.5.7 Gcc_except_table Section Defining Directive (`.gcc_except_table`)

List of gcc_except_table section defining directives

`.gcc_except_table`	Declares the `.gcc_except_table` section in which the table data for exception handling is located.

Instruction format

```
.section .gcc_except_table,"aw",@progbits
.section .gcc_except_table
```

Description

Declares the start of a `.gcc_except_table` section. Statements following this instruction are assembled as those to be mapped in the `.gcc_except_table` section, until another section is declared.

Note that "`aw`" is the flag for the `.section` directive command that specifies data locations and enables write operations, for which a section with read/write attributes is created.

`@progbits` indicates that this is the section containing actual data.

This section is generated by the C/C++ compiler when exception processing is used in C++, and not the sections created exclusively by the user.

8.5.8 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` *<word data>* `[`**`,`***<word data>* ... **`,`***<word data>*`]`

`.short` *<halfword data>* `[`**`,`***<halfword data>* ... **`,`***<halfword data>*`]`

`.byte` *<byte data>* `[`**`,`***<byte data>* ... **`,`***<byte data>*`]`

`.ascii` **`"`***<character string>***`"`** `[`**`,"`***<character string>***`"`** ... **`,"`***<character string>***`"`**`]`

`.space` *<length>* `[`**`,`***<byte data>*`]`

<word data>	4-byte data (`0x0`–`0xffffffff`)
<halfword data>	2-byte data (`0x0`–`0xffff`)
<byte data>	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 word, half word, or 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 `.long`

(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 *<byte data>*. The area begins from the current address unless it is immediately preceded by the `.align` directive.

If *<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)
```

8.5.9 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.
```

8.5.10 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 = <*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.

8.5.11 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.

8.5.12 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 (32-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 Pseudo-operands

To specify offset addresses from the data area start address, the **as** assembler provides the pseudo-operands.
For the data area, refer to Section 3.7.1, "Data Area".
The following 5 types of pseudo-operands can be used:

List of pseudo-operands

`doff_hi(`*<Symbol>*`)`	Acquires the 13 high-order bits of an offset from the default data area start address (`%r15`; `__dp`).
`doff_lo(`*<Symbol>*`)`	Acquires the 13 low-order bits of an offset from the default data area start address (`%r15`; `__dp`).

The following pseudo-operands are for the C33 ADV Core and can be used when the `-mc33adv` option is specified.

`dpoff_h(`*<Symbol>*`)`	Acquires the 13 high-order bits of an offset from the default data area start address (`%dp`; `__dp`).
`dpoff_m(`*<Symbol>*`)`	Acquires the 13 mid-order bits of an offset from the default data area start address (`%dp`; `__dp`).
`dpoff_l(`*<Symbol>*`)`	Acquires the 6 low-order bits of an offset from the default data area start address (`%dp`; `__dp`).

Description

When accessing data in a data area, the base address (data area pointer) of the data area is set to a register (`%r15`, `%dp`) and the offset from the base address to the data is specified with the `ext` instruction. The pseudo-operands can be used as the operand of the `ext` instruction in this case.

Examples:

```
ext   doff_hi(FOO)
ext   doff_lo(FOO)
ld.w  %r0,[%r15]       ← Functions as "ld.w %r0,[%r15 + (FOO address - __dp)]".
```

The offset value acquired by these pseudo-operands are calculated as follows:

<Relocated address of the symbol> - <Data area pointer value (`__*dp`) specified in the linker script file>

Note

- The pseudo-operands are effective only on the defined symbols. If a pseudo-operand is applied to a numeric value, an error will result.
- The symbols must be located at a higher address than the data area pointer (`__dp`). An error will result if the data area pointer value is higher than the symbol address during linkage.
- When using pseudo-operands, the data area start address must be set to the corresponding data area pointers.
  `%r15 = __dp`

8.7 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

immX	Unsigned *X*-bit immediate
signX	Signed *X*-bit immediate
symbol	Symbol to indicate memory address
label	Jump address label
(X:Y)	Bit field from bit *X* to bit *Y*

8.7.1 Arithmetic Operation Instructions

Types and functions of extended instructions

Extended instruction	Function	Expansion
xadd %rd,*imm32*	%rd ← %rd+*imm32*	(1)
xsub %rd,*imm32*	%rd ← %rd-*imm32*	(1)

These extended instructions allow a 32-bit immediate to be specified directly in an add or subtract operation.

Basic instructions after expansion

xadd	Expanded into the `add` instruction
xsub	Expanded into the `sub` instruction

Expansion format

(1) x*OP* %rd,*imm32* (*OP* = add, sub)

Example: xadd %rd,*imm32*

imm32 ≤ 0x3f	0x3f < *imm32* ≤ 0x7ffff	*imm32* > 0x7ffff
add %rd,*imm32*(5:0)	ext *imm32*(18:6) add %rd,*imm32*(5:0)	ext *imm32*(31:19) ext *imm32*(18:6) add %rd,*imm32*(5:0)

8.7.2 Comparison Instructions

Types and functions of extended instructions

Extended instruction	Function	Expansion
xcmp %rd,*sign32*	%rd-*sign32* (Sets/resets C, V, Z and N flags in PSR)	(1)

This extended instruction allows you to compare a general-purpose register and a signed 32-bit immediate.

Basic instruction after expansion

xcmp Expanded into the `cmp` instruction

Expansion format

(1) xcmp %rd,*sign32*

-32 ≤ *sign32* ≤ 31	-262144 ≤ *sign32* < -32 or 31 < *sign32* ≤ 262143	*sign32* < -262144 or 262143 < *sign32*
cmp %rd,*sign32*(5:0)	ext *sign32*(18:6) cmp %rd,*sign32*(5:0)	ext *sign32*(31:19) ext *sign32*(18:6) cmp %rd,*sign32*(5:0)

8.7.3 Logic Operation Instructions

Types and functions of extended instructions

Extended instruction	Function	Expansion
`xand %rd,`*sign32*	`%rd ← %rd &` *sign32*	(1)
`xoor %rd,`*sign32*	`%rd ← %rd |` *sign32*	(1)
`xxor %rd,`*sign32*	`%rd ← %rd ^` *sign32*	(1)
`xnot %rd,`*sign32*	`%rd ← !`*sign32*	(1)

These extended instructions allow a signed 32-bit immediate to be specified directly in a logical operation.

Basic instructions after expansion

xand	Expanded into the `and` instruction
xoor	Expanded into the `or` instruction
xxor	Expanded into the `xor` instruction
xnot	Expanded into the `not` instruction

Expansion formats

(1) x*OP* %rd,*sign32* (*OP* = `and`, `oor`, `xor`, `not`)

Example: `xand %rd,`*sign32*

-32 ≤ *sign32* ≤ 31	-262144 ≤ *sign32* < -32 or 31 < *sign32* ≤ 262143	*sign32* < -262144 or 262143 < *sign32*

`and %rd,`*sign32*`(5:0)`	`ext `*sign32*`(18:6)`
	`and %rd,`*sign32*`(5:0)`
		`ext `*sign32*`(31:19)`
		`ext `*sign32*`(18:6)`
		`and %rd,`*sign32*`(5:0)`

8.7.4 Shift & Rotate Instructions

Types and functions of extended instructions

Extended instruction	Function	Expansion
`xsrl    %rd,`*imm5*	Logical shift to right	(1)
`xsll    %rd,`*imm5*	Logical shift to left	(1)
`xsra    %rd,`*imm5*	Arithmetic shift to right	(1)
`xsla    %rd,`*imm5*	Arithmetic shift to left	(1)
`xrr     %rd,`*imm5*	Rotation to right	(1)
`xrl     %rd,`*imm5*	Rotation to left	(1)

These extended instructions allow a shift or rotate operation to be performed in up to 31 bits.

Basic instructions after expansion

`xsrl`	Expanded into the `srl` instruction
`xsll`	Expanded into the `sll` instruction
`xsra`	Expanded into the `sra` instruction
`xsla`	Expanded into the `sla` instruction
`xrr`	Expanded into the `rr` instruction
`xrl`	Expanded into the `rl` instruction

Expansion formats

(1) `x`*OP* `%rd,`*imm5* (*OP* = `srl`, `sll`, `sra`, `sla`, `rr`, `rl`)

Example: `xsrl   %rd,`*imm5*

imm5 ≤ 8	8 < *imm5* < 16	*imm5* = 16
`srl     %rd,`*imm5*`(3:0)`	`srl     %rd,0x8` `srl     %rd,`*imm5*`(2:0)`	`srl     %rd,0x8` `srl     %rd,0x8`
16 < *imm5* ≤ 24	***imm5* > 24**	

`srl     %rd,0x8`
`srl     %rd,0x8`
`srl     %rd,`*imm5*`(3:0)`
	`srl     %rd,0x8`
	`srl     %rd,0x8`
	`srl     %rd,0x8`
	`srl     %rd,`*imm5*`(2:0)`	

When the `-mc33adv` option (for C33 ADV Core) is specified, these extended instructions are expanded into one basic instruction.

Example: `xsrl   %rd,`*imm5* → `srl   %rd,`*imm5*

8.7.5 Data Transfer Instructions (between Stack and Register)

Types and functions of extended instructions

Extended instruction	Function	Expansion
`xld.b  %rd,[%sp+`*imm32*`]`	`%rd ← B[%sp+`*imm32*`]` (with sign extension)	(1)
`xld.ub %rd,[%sp+`*imm32*`]`	`%rd ← B[%sp+`*imm32*`]` (with zero extension)	(1)
`xld.h  %rd,[%sp+`*imm32*`]`	`%rd ← H[%sp+`*imm32*`]` (with sign extension)	(2)
`xld.uh %rd,[%sp+`*imm32*`]`	`%rd ← H[%sp+`*imm32*`]` (with zero extension)	(2)
`xld.w  %rd,[%sp+`*imm32*`]`	`%rd ← W[%sp+`*imm32*`]`	(3)
`xld.b  [%sp+`*imm32*`],%rs`	`B[%sp+`*imm32*`] ← %rs(7:0)`	(1)
`xld.h  [%sp+`*imm32*`],%rs`	`H[%sp+`*imm32*`] ← %rs(15:0)`	(2)
`xld.w  [%sp+`*imm32*`],%rs`	`W[%sp+`*imm32*`] ← %rs`	(3)

These extended instructions allow you to directly specify a displacement of up to 32 bits. Specification of `imm32` can be omitted.

Basic instructions after expansion

`xld.b`	Expanded into the `ld.b` instruction
`xld.ub`	Expanded into the `ld.ub` instruction
`xld.h`	Expanded into the `ld.h` instruction
`xld.uh`	Expanded into the `ld.uh` instruction
`xld.w`	Expanded into the `ld.w` instruction

Expansion formats

If `imm32` is omitted, the **as** assembler assumes that `[%sp+0x0]` is specified as it expands the instruction.

(1) Byte data transfer (`xld.b`, `xld.ub`)

Example: `xld.b  %rd,[%sp+`*imm32*`]`

imm32 ≤ 0x3f	0x3f < *imm32* ≤ 0x7ffff	*imm32* > 0x7ffff

`ld.b  %rd,[%sp+imm32(5:0)]`	`ext   imm32(18:6)`
	`ld.b  %rd,[%sp+imm32(5:0)]`
		`ext   imm32(31:19)`
		`ext   imm32(18:6)`
		`ld.b  %rd,[%sp+imm32(5:0)]`

(2) Half word data transfer (`xld.h`, `xld.uh`)

Example: `xld.h  %rd,[%sp+`*imm32*`]`

imm32 ≤ 0x7f	0x7f < *imm32* ≤ 0x7ffff	*imm32* > 0x7ffff

`ld.h  %rd,[%sp+imm32(6:1)]`	`ext   imm32(18:6)`
	`ld.h  %rd,[%sp+imm32(5:0)]`
		`ext   imm32(31:19)`
		`ext   imm32(18:6)`
		`ld.h  %rd,[%sp+imm32(5:0)]`

(3) Word data transfer (`xld.w`)

Example: `xld.w  %rd,[%sp+`*imm32*`]`

imm32 ≤ 0xff	0xff < *imm32* ≤ 0x7ffff	*imm32* > 0x7ffff

`ld.w  %rd,[%sp+imm32(7:2)]`	`ext   imm32(18:6)`
	`ld.w  %rd,[%sp+imm32(5:0)]`
		`ext   imm32(31:19)`
		`ext   imm32(18:6)`
		`ld.w  %rd,[%sp+imm32(5:0)]`

8.7.6 Data Transfer Instructions (between Memory and Register)

Types and functions of extended instructions

Extended instruction	Function	Expansion
`xld.b  %rd,[symbol+imm26]`	`%rd ← B[symbol+imm26]` (with sign extension) *	(1)
`xld.ub %rd,[symbol+imm26]`	`%rd ← B[symbol+imm26]` (with zero extension) *	(1)
`xld.h  %rd,[symbol+imm26]`	`%rd ← H[symbol+imm26]` (with sign extension) *	(1)
`xld.uh %rd,[symbol+imm26]`	`%rd ← H[symbol+imm26]` (with zero extension) *	(1)
`xld.w  %rd,[symbol+imm26]`	`%rd ← W[symbol+imm26]` *	(1), (2)
`xld.b  [symbol+imm26],%rs`	`B[symbol+imm26] ← %rs(7:0)` *	(1)
`xld.h  [symbol+imm26],%rs`	`H[symbol+imm26] ← %rs(15:0)` *	(1)
`xld.w  [symbol+imm26],%rs`	`W[symbol+imm26] ← %rs` *	(1)
`xld.b  %rd,[%rb+imm26]`	`%rd ← B[%rb+imm26]` (with sign extension)	(3)
`xld.ub %rd,[%rb+imm26]`	`%rd ← B[%rb+imm26]` (with zero extension)	(3)
`xld.h  %rd,[%rb+imm26]`	`%rd ← H[%rb+imm26]` (with sign extension)	(3)
`xld.uh %rd,[%rb+imm26]`	`%rd ← H[%rb+imm26]` (with zero extension)	(3)
`xld.w  %rd,[%rb+imm26]`	`%rd ← W[%rb+imm26]`	(3)
`xld.b  [%rb+imm26],%rs`	`B[%rb+imm26] ← %rs(7:0)`	(3)
`xld.h  [%rb+imm26],%rs`	`H[%rb+imm26] ← %rs(15:0)`	(3)
`xld.w  [%rb+imm26],%rs`	`W[%rb+imm26] ← %rs`	(3)

∗ In the C33 ADV Core, *imm26* is extended into *imm32*.

These extended instructions allow memory locations to be accessed by specifying the address with a symbol or 32-bit immediate. However, the postincrement function ([] +) cannot be used. Specification of *imm26* (*imm32*) can be omitted.

Basic instructions after expansion

`xld.b`	Expanded into the `ld.b` instruction
`xld.ub`	Expanded into the `ld.ub` instruction
`xld.h`	Expanded into the `ld.h` instruction
`xld.uh`	Expanded into the `ld.uh` instruction
`xld.w`	Expanded into the `ld.w` instruction

Expansion formats

(1) **`xld.*   %rd,[symbol+imm26]`** (* = b, ub, h, uh, w)
`xld.*   [symbol+imm26],%rs` (* = b, h, w)

Example: `xld.w   %rd,[symbol+imm26]`

Unconditional
`ext     (symbol+imm26)(25:13)`
`ext     (symbol+imm26)(12:0)`
`ld.w    %rd,[%r15]`

If *imm26* is omitted, the **as** assembler assumes that `[symbol+0x0]` is specified as it expands the instruction.

When the `-medda32` option is specified

`xld.* %rd,[symbol+imm26]`	`xld.* [symbol+imm26],%rs` (`%rs` ≠ `%r0`)	`xld.* [symbol+imm26],%rs` (`%rs` = `%r0`)

`ext     (symbol+imm26)@h`
`ext     (symbol+imm26)@m`
`ld.w    %rd,(symbol+imm26)@l`
`ld.*    %rd,[%rd]`
	`pushn   %r0`
	`ext     (symbol+imm26)@h`
	`ext     (symbol+imm26)@m`
	`ld.w    %r0,(symbol+imm26)@l`
	`ld.*    [%r0],%rs`
	`popn    %r0`
		`pushn   %r1` (or `push %r1`)*1
		`ext     (symbol+imm26)@h`
		`ext     (symbol+imm26)@m`
		`ld.w    %r1,(symbol+imm26)@l`
		`ld.*    [%r1],%r0`
		`popn    %r1` (or `pop %r1`)*1

∗1 When the `-mc33adv` or `-mc33pe` option is specified

(2) `xld.w %rd,[symbol+imm32]`

When the `-mc33adv` option is specified

Unconditional

`ext (symbol+imm32)(31:19)`
`ext (symbol+imm32)(18:6)`
`ld.w %rd,[%dp+(symbol+imm32)(5:0)]`

(3) `xld.* %rd,[%rb+imm26]` (* = b, ub, h, uh, w)

`xld.* [%rb+imm26],%rs` (* = b, h, w)

Example: `xld.w %rd,[%rb+imm26]`

imm26 = 0	1 ≤ *imm26* ≤ 0x1fff	*imm26* > 0x1fff

`ld.w %rd,[%rb]`	`ext imm26(12:0)`
	`ld.w %rd,[%rb]`
		`ext imm26(25:13)`
		`ext imm26(12:0)`
		`ld.w %rd,[%rb]`

If *imm26* is omitted, the **as** assembler assumes that `[%rb+0x0]` is specified as it expands the instruction.

8.7.7 Immediate Data Load Instructions

Types and functions of extended instructions

Extended instruction	Function	Expansion
xld.w %rd,*symbol±imm32*	%rd ← *symbol±imm32*	(1)
xld.w %rd,*sign32*	%rd ← *sign32*	(2)

* "*symbol±imm32*" means that "*symbol+imm32*" and "*symbol-imm32*" can be specified.

These extended instructions allow a 32-bit immediate to be loaded directly into a general-purpose register. A symbol also can be used for immediate specification.

Basic instruction after expansion

xld.w Expanded into the `ld.w` instruction

Expansion formats

(1) xld.w %rd,*symbol+imm32*
xld.w %rd,*symbol-imm32*

Example: xld.w %rd,*symbol+imm32*

Unconditional
ext (*symbol+imm32*)@h
ext (*symbol+imm32*)@m
ld.w %rd,(*symbol+imm32*)@l

Specification of *imm32* can be omitted. If *imm32* is omitted, the **as** assembler assumes that "*symbol*+0x0" is specified as it expands the instruction.

(2) xld.w %rd,*sign32*

-32 ≤ *sign32* ≤ 31	-262144 ≤ *sign32* < -32 or 31 < *sign32* ≤ 262143	*sign32* < -262144 or 262143 < *sign32*
ld.w %rd,*sign32*(5:0)	ext *sign32*(18:6) ld.w %rd,*sign32*(5:0)	ext *sign32*(31:19) ext *sign32*(18:6) ld.w %rd,*sign32*(5:0)

8.7.8 Bit Operation Instructions

Types and functions of extended instructions

Extended instruction	Function	Expansion
`xbtst [symbol+imm26],imm3`	`B[symbol+imm26]` bit test *	(1)
`xbclr [symbol+imm26],imm3`	`B[symbol+imm26]` bit clear *	(1)
`xbset [symbol+imm26],imm3`	`B[symbol+imm26]` bit set *	(1)
`xbnot [symbol+imm26],imm3`	`B[symbol+imm26]` bit negation *	(1)
`xbtst [%rb+imm26],imm3`	`B[%rb+imm26]` bit test	(2)
`xbclr [%rb+imm26],imm3`	`B[%rb+imm26]` bit clear	(2)
`xbset [%rb+imm26],imm3`	`B[%rb+imm26]` bit set	(2)
`xbnot [%rb+imm26],imm3`	`B[%rb+imm26]` bit negation	(2)

* In the C33 ADV Core, `imm26` is extended into `imm32`.

These extended instructions allow a memory address for manipulating bits to be specified with a symbol and/or a displacement. Specification of `imm26` (`imm32`) can be omitted.

Basic instructions after expansion

xbtst	Expanded into the `btst` instruction
xbclr	Expanded into the `bclr` instruction
xbset	Expanded into the `bset` instruction
xbnot	Expanded into the `bnot` instruction

Expansion formats

(1) `xOP [symbol+imm26],imm3` (*OP* = `btst`, `bclr`, `bset`, `bnot`)

Example: `xbtst [symbol+imm26],imm3`

Unconditional

`ext (symbol+imm26)(25:13)`
`ext (symbol+imm26)(12:0)`
`btst [%r15],imm3`

If `imm26` is omitted, the **as** assembler assumes that `[symbol+0x0]` is specified as it expands the instruction.

When the `-medda32` option is specified

Example: `xbtst [symbol+imm26],imm3`

Unconditional

`pushn %r0`
`ext (symbol+imm26)@h`
`ext (symbol+imm26)@m`
`ld.w %r0,(symbol+imm26)@l`
`btst [%r0],imm3`
`popn %r0`

(2) `xOP [%rb+imm26],imm3` (*OP* = `btst`, `bclr`, `bset`, `bnot`)

Example: `xbtst [%rb+imm26],imm3`

`imm26` = 0	1 ≤ `imm26` ≤ 0x1fff	`imm26` > 0x1fff

`btst %rd,[%rb]`	`ext imm26(12:0)`
	`btst %rd,[%rb]`
		`ext imm26(25:13)`
		`ext imm26(12:0)`
		`btst %rd,[%rb]`

If `imm26` is omitted, the **as** assembler assumes that `[%rb+0x0]` is specified as it expands the instruction.

8.7.9 Branch Instructions

Types and functions of extended instructions

Extended instruction	Function	Expansion
scall *label+imm32*	Subroutine call	(1)
scall.d *label+imm32*	Subroutine call (with delayed branch operation)	(1)
sjp *label+imm32*	Unconditional jump	(1)
sjp.d *label+imm32*	Unconditional jump (with delayed branch operation)	(1)
sjreq *label+imm32*	Conditional jump	(1)
sjreq.d *label+imm32*	Conditional jump (with delayed branch operation)	(1)
sjrne *label+imm32*	Conditional jump	(1)
sjrne.d *label+imm32*	Conditional jump (with delayed branch operation)	(1)
sjrgt *label+imm32*	Conditional jump	(1)
sjrgt.d *label+imm32*	Conditional jump (with delayed branch operation)	(1)
sjrge *label+imm32*	Conditional jump	(1)
sjrge.d *label+imm32*	Conditional jump (with delayed branch operation)	(1)
sjrlt *label+imm32*	Conditional jump	(1)
sjrlt.d *label+imm32*	Conditional jump (with delayed branch operation)	(1)
sjrle *label+imm32*	Conditional jump	(1)
sjrle.d *label+imm32*	Conditional jump (with delayed branch operation)	(1)
sjrugt *label+imm32*	Conditional jump	(1)
sjrugt.d *label+imm32*	Conditional jump (with delayed branch operation)	(1)
sjruge *label+imm32*	Conditional jump	(1)
sjruge.d *label+imm32*	Conditional jump (with delayed branch operation)	(1)
sjrult *label+imm32*	Conditional jump	(1)
sjrult.d *label+imm32*	Conditional jump (with delayed branch operation)	(1)
sjrule *label+imm32*	Conditional jump	(1)
sjrule.d *label+imm32*	Conditional jump (with delayed branch operation)	(1)
scall *sign22*	Subroutine call	(2)
scall.d *sign22*	Subroutine call (with delayed branch operation)	(2)
sjp *sign22*	Unconditional jump	(2)
sjp.d *sign22*	Unconditional jump (with delayed branch operation)	(2)
sjreq *sign22*	Conditional jump	(2)
sjreq.d *sign22*	Conditional jump (with delayed branch operation)	(2)
sjrne *sign22*	Conditional jump	(2)
sjrne.d *sign22*	Conditional jump (with delayed branch operation)	(2)
sjrgt *sign22*	Conditional jump	(2)
sjrgt.d *sign22*	Conditional jump (with delayed branch operation)	(2)
sjrge *sign22*	Conditional jump	(2)
sjrge.d *sign22*	Conditional jump (with delayed branch operation)	(2)
sjrlt *sign22*	Conditional jump	(2)
sjrlt.d *sign22*	Conditional jump (with delayed branch operation)	(2)
sjrle *sign22*	Conditional jump	(2)
sjrle.d *sign22*	Conditional jump (with delayed branch operation)	(2)
sjrugt *sign22*	Conditional jump	(2)
sjrugt.d *sign22*	Conditional jump (with delayed branch operation)	(2)
sjruge *sign22*	Conditional jump	(2)
sjruge.d *sign22*	Conditional jump (with delayed branch operation)	(2)
sjrult *sign22*	Conditional jump	(2)
sjrult.d *sign22*	Conditional jump (with delayed branch operation)	(2)
sjrule *sign22*	Conditional jump	(2)
sjrule.d *sign22*	Conditional jump (with delayed branch operation)	(2)

Extended instruction	Function	Expansion
`xcall` *label+imm32*	Subroutine call	(3)
`xcall.d` *label+imm32*	Subroutine call (with delayed branch operation)	(3)
`xjp` *label+imm32*	Unconditional jump	(3)
`xjp.d` *label+imm32*	Unconditional jump (with delayed branch operation)	(3)
`xjreq` *label+imm32*	Conditional jump	(3)
`xjreq.d` *label+imm32*	Conditional jump (with delayed branch operation)	(3)
`xjrne` *label+imm32*	Conditional jump	(3)
`xjrne.d` *label+imm32*	Conditional jump (with delayed branch operation)	(3)
`xjrgt` *label+imm32*	Conditional jump	(3)
`xjrgt.d` *label+imm32*	Conditional jump (with delayed branch operation)	(3)
`xjrge` *label+imm32*	Conditional jump	(3)
`xjrge.d` *label+imm32*	Conditional jump (with delayed branch operation)	(3)
`xjrlt` *label+imm32*	Conditional jump	(3)
`xjrlt.d` *label+imm32*	Conditional jump (with delayed branch operation)	(3)
`xjrle` *label+imm32*	Conditional jump	(3)
`xjrle.d` *label+imm32*	Conditional jump (with delayed branch operation)	(3)
`xjrugt` *label+imm32*	Conditional jump	(3)
`xjrugt.d` *label+imm32*	Conditional jump (with delayed branch operation)	(3)
`xjruge` *label+imm32*	Conditional jump	(3)
`xjruge.d` *label+imm32*	Conditional jump (with delayed branch operation)	(3)
`xjrult` *label+imm32*	Conditional jump	(3)
`xjrult.d` *label+imm32*	Conditional jump (with delayed branch operation)	(3)
`xjrule` *label+imm32*	Conditional jump	(3)
`xjrule.d` *label+imm32*	Conditional jump (with delayed branch operation)	(3)
`xcall` *sign32*	Subroutine call	(4)
`xcall.d` *sign32*	Subroutine call (with delayed branch operation)	(4)
`xjp` *sign32*	Unconditional jump	(4)
`xjp.d` *sign32*	Unconditional jump (with delayed branch operation)	(4)
`xjreq` *sign32*	Conditional jump	(4)
`xjreq.d` *sign32*	Conditional jump (with delayed branch operation)	(4)
`xjrne` *sign32*	Conditional jump	(4)
`xjrne.d` *sign32*	Conditional jump (with delayed branch operation)	(4)
`xjrgt` *sign32*	Conditional jump	(4)
`xjrgt.d` *sign32*	Conditional jump (with delayed branch operation)	(4)
`xjrge` *sign32*	Conditional jump	(4)
`xjrge.d` *sign32*	Conditional jump (with delayed branch operation)	(4)
`xjrlt` *sign32*	Conditional jump	(4)
`xjrlt.d` *sign32*	Conditional jump (with delayed branch operation)	(4)
`xjrle` *sign32*	Conditional jump	(4)
`xjrle.d` *sign32*	Conditional jump (with delayed branch operation)	(4)
`xjrugt` *sign32*	Conditional jump	(4)
`xjrugt.d` *sign32*	Conditional jump (with delayed branch operation)	(4)
`xjruge` *sign32*	Conditional jump	(4)
`xjruge.d` *sign32*	Conditional jump (with delayed branch operation)	(4)
`xjrult` *sign32*	Conditional jump	(4)
`xjrult.d` *sign32*	Conditional jump (with delayed branch operation)	(4)
`xjrule` *sign32*	Conditional jump	(4)
`xjrule.d` *sign32*	Conditional jump (with delayed branch operation)	(4)

These extended instructions allow a branch destination to be specified using a label with displacement included or a signed 22 or 32-bit immediate. The branch conditions of these conditional jump instructions are the same as those of the basic instructions.

Basic instructions after expansion

`scall/scall.d/xcall/xcall.d`	Expanded into the `call/call.d` instruction
`sjp/sjp.d/xjp/xjp.d`	Expanded into the `jp/jp.d` instruction
`sjreq/sjreq.d/xjreq/xjreq.d`	Expanded into the `jreq/jreq.d` instruction
`sjrne/sjrne.d/xjrne/xjrne.d`	Expanded into the `jrne/jrne.d` instruction
`sjrgt/sjrgt.d/xjrgt/xjrgt.d`	Expanded into the `jrgt/jrgt.d` instruction
`sjrge/sjrge.d/xjrge/xjrge.d`	Expanded into the `jrge/jrge.d` instruction
`sjrlt/sjrlt.d/xjrlt/xjrlt.d`	Expanded into the `jrlt/jrlt.d` instruction
`sjrle/sjrle.d/xjrle/xjrle.d`	Expanded into the `jrle/jrle.d` instruction
`sjrugt/sjrugt.d/xjrugt/xjrugt.d`	Expanded into the `jrugt/jrugt.d` instruction
`sjruge/sjruge.d/xjruge/xjruge.d`	Expanded into the `jruge/jruge.d` instruction
`sjrult/sjrult.d/xjrult/xjrult.d`	Expanded into the `jrult/jrult.d` instruction
`sjrule/sjrule.d/xjrule/xjrule.d`	Expanded into the `jrule/jrule.d` instruction

Expansion formats

(1) **`s`*`OP`*** ***`label+imm32`*** (*`OP`* = `call`, `call.d`, `jp`, `jp.d`, `jr*`, `jr*.d`)

Example: `scall` *`label+imm32`*

Unconditional
`ext    (label+imm32)@rm` `call   (label+imm32)@rl`

Specification of *`imm32`* can be omitted. If *`imm32`* is omitted, the **as** assembler assumes that "*`label`*`+0x0`" is specified as it expands the instruction.

(2) **`s`*`OP`*** ***`sign22`*** (*`OP`* = `call`, `call.d`, `jp`, `jp.d`, `jr*`, `jr*.d`)

Example: `scall` *`sign22`*

-256 ≤ *`sign22`* ≤ 254	-2097152 ≤ *`sign22`* < -256 or 254 < *`sign22`* ≤ 2097150
`call   sign22(8:1)`	`ext    sign22(21:9)` `call   sign22(8:1)`

(3) **`x`*`OP`*** ***`label+imm32`*** (*`OP`* = `call`, `call.d`, `jp`, `jp.d`, `jr*`, `jr*.d`)

Example: `xcall` *`label+imm32`*

Unconditional

`ext    (label+imm32)@rh`
`ext    (label+imm32)@rm`
`call   (label+imm32)@rl`

Specification of *`imm32`* can be omitted. If *`imm32`* is omitted, the **as** assembler assumes that "*`label`*`+0x0`" is specified as it expands the instruction.

(4) **`x`*`OP`*** ***`sign32`*** (*`OP`* = `call`, `call.d`, `jp`, `jp.d`, `jr*`, `jr*.d`)

Example: `xcall` *`sign32`*

-256 ≤ *`sign32`* ≤ 254	-2097152 ≤ *`sign32`* < -256 or 254 < *`sign32`* ≤ 2097150	*`sign32`* < -2097152 or 2097150 < *`sign32`*

`call   sign32(8:1)`	`ext    sign32(21:9)`
	`call   sign32(8:1)`
		`ext    sign32(31:22)<<0x03`
		`ext    sign32(21:9)`
		`call   sign32(8:1)`

8.7.10 C33 ADV Core Data Transfer Instructions (between Memory and Register)

The extended instructions described in this section are provided only for C33 ADV Core and can be used when the `-mc33adv` option is specified.

Types and functions of extended instructions

Extended instruction	Function	Expansion
ald.b %rd,[*symbol+imm19*]	%rd ← B[*symbol+imm19*] (with sign extension)	(1)
ald.ub %rd,[*symbol+imm19*]	%rd ← B[*symbol+imm19*] (with zero extension)	(1)
ald.h %rd,[*symbol+imm19*]	%rd ← H[*symbol+imm19*] (with sign extension)	(1)
ald.uh %rd,[*symbol+imm19*]	%rd ← H[*symbol+imm19*] (with zero extension)	(1)
ald.w %rd,[*symbol+imm19*]	%rd ← W[*symbol+imm19*]	(1)
ald.b [*symbol+imm19*],%rs	B[*symbol+imm19*] ← %rs(7:0)	(1)
ald.h [*symbol+imm19*],%rs	H[*symbol+imm19*] ← %rs(15:0)	(1)
ald.w [*symbol+imm19*],%rs	W[*symbol+imm19*] ← %rs	(1)

These extended instructions allow memory locations to be accessed by specifying the address with a symbol and 32-bit immediate. However, the postincrement function (`[]+`) cannot be used. Specification of *`imm19`* can be omitted.

Basic instructions after expansion

ald.b	Expanded into the `ld.b` instruction
ald.ub	Expanded into the `ld.ub` instruction
ald.h	Expanded into the `ld.h` instruction
ald.uh	Expanded into the `ld.uh` instruction
ald.w	Expanded into the `ld.w` instruction

Expansion formats

If *`imm19`* is omitted, the **as** assembler assumes that `[`*`symbol`*`+0x0]` is specified as it expands the instruction.

(1) ald.* %rd,[*symbol+imm19*] (* = b, ub, h, uh, w)
ald.* [*symbol+imm19*],%rs (* = b, h, w)

Example: ald.w %rd,[*symbol+imm19*]

Unconditional
ext (*symbol+imm19*)(18:6)
ld.w %rd,[%dp+(*symbol+imm19*)(5:0)]

When the `-medda32` option is specified

ald.* %rd,[*symbol+imm19*]	ald.* [*symbol+imm19*],%rs (%rs ≠ %r0)	ald.* [*symbol+imm19*],%rs (%rs = %r0)
ext (*symbol+imm19*)@m ld.w %rd,(*symbol+imm19*)@l ld.* %rd,[%rd]	pushn %r0 ext (*symbol+imm19*)@m ld.w %r0,(*symbol+imm19*)@l ld.* [%r0],%rs popn %r0	pushn %r1 (or push %r1)*1 ext (*symbol+imm19*)@m ld.w %r1,(*symbol+imm19*)@l ld.* [%r1],%r0 popn %r1 (or pop %r1)*1

*1 When the `-mc33adv` or `-mc33pe` option is specified

8.7.11 C33 ADV Core Data Transfer Instructions (between `%dp` Area and Register)

The extended instructions described in this section are provided only for C33 ADV Core and can be used when the `-mc33adv` option is specified.

Types and functions of extended instructions

Extended instruction	Function	Expansion
`xld.b  %rd,[%dp+imm32]`	`%rd ← B[%dp+imm32]` (with sign extension)	(1)
`xld.ub %rd,[%dp+imm32]`	`%rd ← B[%dp+imm32]` (with zero extension)	(1)
`xld.h  %rd,[%dp+imm32]`	`%rd ← H[%dp+imm32]` (with sign extension)	(2)
`xld.uh %rd,[%dp+imm32]`	`%rd ← H[%dp+imm32]` (with zero extension)	(2)
`xld.w  %rd,[%dp+imm32]`	`%rd ← W[%dp+imm32]`	(3)
`xld.b  [%dp+imm32],%rs`	`B[%dp+imm32] ← %rs(7:0)`	(1)
`xld.h  [%dp+imm32],%rs`	`H[%dp+imm32] ← %rs(15:0)`	(2)
`xld.w  [%dp+imm32],%rs`	`W[%dp+imm32] ← %rs`	(3)

These extended instructions allow memory locations to be accessed by specifying the address with a 32-bit immediate. However, the postincrement function (`[]+`) cannot be used. Specification of *imm32* can be omitted.

Basic instructions after expansion

xld.b	Expanded into the `ld.b` instruction
xld.ub	Expanded into the `ld.ub` instruction
xld.h	Expanded into the `ld.h` instruction
xld.uh	Expanded into the `ld.uh` instruction
xld.w	Expanded into the `ld.w` instruction

Expansion formats

If *imm32* is omitted, the **as** assembler assumes that `[%dp+0x0]` is specified as it expands the instruction.

(1) Byte data transfer (`xld.b`, `xld.ub`)

Example: `xld.b   %rd,[%dp+imm32]`

imm32 ≤ 0x3f	0x3f < *imm32* ≤ 0x7ffff	*imm32* > 0x7ffff

`ld.b   %rd,[%dp+imm32(5:0)]`	`ext    imm32(18:6)`
	`ld.b   %rd,[%dp+imm32(5:0)]`
		`ext    imm32(31:19)`
		`ext    imm32(18:6)`
		`ld.b   %rd,[%dp+imm32(5:0)]`

(2) Half word data transfer (`xld.h`, `xld.uh`)

Example: `xld.h   %rd,[%dp+imm32]`

imm32 ≤ 0x7f	0x7f < *imm32* ≤ 0x7ffff	*imm32* > 0x7ffff

`ld.h   %rd,[%dp+imm32(6:1)]`	`ext    imm32(18:6)`
	`ld.h   %rd,[%dp+imm32(5:0)]`
		`ext    imm32(31:19)`
		`ext    imm32(18:6)`
		`ld.h   %rd,[%dp+imm32(5:0)]`

(3) Word data transfer (`xld.w`)

Example: `xld.w   %rd,[%dp+imm32]`

imm32 ≤ 0xff	0xff < *imm32* ≤ 0x7ffff	*imm32* > 0x7ffff

`ld.w   %rd,[%dp+imm32(7:2)]`	`ext    imm32(18:6)`
	`ld.w   %rd,[%dp+imm32(5:0)]`
		`ext    imm32(31:19)`
		`ext    imm32(18:6)`
		`ld.w   %rd,[%dp+imm32(5:0)]`

8.8 Optimization of Extended Instructions

The C/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 `-mc33_ext` option is used for the optimization to remove unnecessary `ext` instructions.

`-mc33_ext` option

`-mc33_ext` *filename*`.dump`

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`
allobjects`.dump`	This file contains the global information of all object files after assembling in the first pass has finished. To create the file, execute **objdump** with all object files specified as follows: `objdump -t *.o >> allobjects.dump` The file name extension must be "`.dump`". This file is used to remove global symbols that may be duplicated in objects from optimization.

Example:
```
objdump -t test.elf > test.dump
objdump -t *.o >> allobjects.dump
as -mc33_ext test.dump allobjects.dump -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 dump file specified with the `-mc33_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 `-mc33_ext` option)
3. Link (required for generating a dump file)
4. Create a dump file from the elf file using **objdump**.
5. Create a dump file (allobjects.dump) from the all object files using **objdump**.
6. Reassemble (with the `-mc33_ext` option) * Specify the same input/output files as Step 2.

The makefile created by the **IDE** contains the procedure above when two-pass make process is specified.

Optimize patterns

The optimization process is classified under five patterns.

Optimization 1: for data transfer instructions (memory to register)

Example:

```
sub:
     ret
main:
          ...
     xld.w   %r0,[sub]
          ...
     ret
```

When **`-medda32`** is specified

Expansion format

```
xld.*  %r0,[sub]                  (* = b, ub, h, uh or w)
     ↓
ext     <high-order 13 bits of the symbol>
ext     <mid-order 13 bits of the symbol>
ld.w    %rd,<low-order 6 bits of the symbol>
ld.*    %rd,[%rd]
```

The assembler obtains the address of the symbol (`sub`) from the dump file to determine the number of `ext` instructions to be used.

Symbol address = 0 to 0x1f: Expanded with no `ext` used
Symbol address = 0x20 to 0x3ffff: Expanded with one `ext` used
Symbol address = 0x40000 to 0xffffff: Expanded with two `ext` used

When **`-medda32`** is not specified

Expansion format 1: for C33 STD or C33 PE

```
xld.w  %r0,[sub]
     ↓
ext     imm13
ext     imm13
ld.w    %r0,[%r15]
```

The assembler determines the number of `ext` instructions to be used according to the address of the symbol (`sub`) and the data pointer value set in the `%r15` register.

Symbol address - data pointer = 0 to 0x1fff: Expanded with one `ext` used
Symbol address - data pointer = 0x2000 to 0x3ffffff: Expanded with two `ext` used

Expansion format 2: for S1C33401

```
xld.w  %r0,[sub]
     ↓
ext     imm13
ext     imm13
ld.w    %r0,[%dp+imm6]
```

The assembler determines the number of `ext` instructions to be used according to the address of the symbol (`sub`) and the data pointer value set in the `%dp` register.

Symbol address - data pointer = 0 to 0x7ffff: Expanded with one `ext` used
Symbol address - data pointer = 0x80000 to 0xfffffff: Expanded with two `ext` used

Optimization 2: for data transfer instructions (register to memory)

Example:

```
label:
     .skip   4
main:
          ...
     xld.w   [label],%r0
          ...
     ret
```

When `-medda32` is specified

Expansion format

```
xld.*   [label],%r0                    (* = b, h or w)
     ↓
pushn   %r1
ext     <high-order 13 bits of the symbol>
ext     <mid-order 13 bits of the symbol>
ld.w    %r1, <low-order 6 bits of the symbol>
ld.*    [%r1],%rs
popn    %r1
```

The assembler obtains the address of the symbol (`label`) from the dump file to determine the number of `ext` instructions to be used.

Symbol address = 0 to 0x1f:	Expanded with no `ext` used
Symbol address = 0x20 to 0x3ffff:	Expanded with one `ext` used
Symbol address = 0x40000 to 0xffffff:	Expanded with two `ext` used

When `-medda32` is not specified

Expansion format 1: for C33 STD or C33 PE

```
xld.w   [label],%r0
     ↓
ext     imm13
ext     imm13
ld.w    [%r15],%r0
```

The assembler determines the number of `ext` instructions to be used according to the address of the symbol (`label`) and the data pointer value set in the `%r15` register.

Symbol address - data pointer = 0 to 0x1fff:	Expanded with one `ext` used
Symbol address - data pointer = 0x2000 to 0x3ffffff:	Expanded with two `ext` used

Expansion format 2: for S1C33401

```
xld.w   [label],%r0
     ↓
ext     imm13
ext     imm13
ld.w    [%dp+imm6],%r0
```

The assembler determines the number of `ext` instructions to be used according to the address of the symbol (`label`) and the data pointer value set in the `%dp` register.

Symbol address - data pointer = 0 to 0x7ffff:	Expanded with one `ext` used
Symbol address - data pointer = 0x80000 to 0xfffffff:	Expanded with two `ext` used

Optimization 3: for bit operation instructions

Example:

```
main:
           ...
      xbtst   [SYMBOL],1
           ...
      ret
```

When `-medda32` is specified

Expansion format

```
xbtst  [SYMBOL],1
     ↓
pushn  %r0
ext    <high-order 13 bits of the symbol>
ext    <mid-order 13 bits of the symbol>
ld.w   %r0, <low-order 6 bits of the symbol>
b*     [%r0],imm3                          (* = tst, clr, set or not)
popn   %r0
```

The assembler obtains the address of the symbol (`SYMBOL`) from the dump file to determine the number of `ext` instructions to be used.

Symbol address = 0 to 0x1f: Expanded with no `ext` used
Symbol address = 0x20 to 0x3ffff: Expanded with one `ext` used
Symbol address = 0x40000 to 0xffffff: Expanded with two `ext` used

When `-medda32` is not specified

Expansion format

```
xbtst  [SYMBOL],1
     ↓
ext    imm13
ext    imm13
b*     [%r15],imm3                         (* = tst, clr, set or not)
```

The assembler determines the number of `ext` instructions to be used according to the address of the symbol (`SYMBOL`) and the data pointer value set in the `%r15` register.

Symbol address - data pointer = 0 to 0x1fff: Expanded with one `ext` used
Symbol address - data pointer = 0x2000 to 0x3ffffff: Expanded with two `ext` used

Optimization 4: for relative branch (`scall`/`xcall`/`sjp`/`xjp`/`sjrxx`/`xjrxx`) instructions

Example:

```
sub:
          ret

main:
                    ...
          xcall   sub              ← ADDR1
                    ...
          ret
```

The assembler obtains the destination address (`sub`) directly from the dump file. The branch address (ADDR1) is not included in the dump file. Therefore, the assembler counts the instructions from the beginning address of the `main` routine that can be obtained from the dump file 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.

Branch distance = -256 to 254: Expanded with no `ext` used
Branch distance = -2,097,152 to -257 or 255 to 2,097,150: Expanded with one `ext` used
Branch distance = -2,147,483,648 to -2,097,153 or 2,097,151 to 2,147,483,647: 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
          xjp   L1      ;Check (L1 - branch address) value
```

Before optimization	After optimization
`nop`	`nop`
`ext  0x1ff8` ← operand 1 (*imm10* << 3)	`jp  0x7e`
`ext  0x1fff` ← operand 2 (*imm13*)	
`jp   0xfd` ← operand 3 (*sign9* = (8:1))	

Optimization 5: when the operand is an expression using a symbol

Example:

```
sub:
          nop
          ret

main:
                    ...
          xld.w    %r0,[sub+0x2]
                    ...
          ret
```

The assembler evaluates the expression in the operand and determines the number of ext instructions to be used. The expansion format in Optimization 1 is applied to the example above.

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, bit 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.
- When executing a build without the `-medda32` option, the settings for data pointer shown below are required. Otherwise, no optimization for extended instructions will be performed.

  1. Specify the data pointer value in the linker script file.
     Write the command shown below in the linker script file.
     ```
     __dp=0xN;
     ```
     This command is required to set the data pointer. In the **IDE**, it can be specified in the [GNU33 Linker Script Settings] page on the project property dialog box.
  2. Set the `__dp` value to the `%r15` and `%dp` registers in the boot routine.
     Substitute the `__dp` symbol value into the `%r15` register.
     In the S1C33401, the same value must be set to the `%dp` register.
     ```
     xld.w  %r15,__dp
     ld.w   %dp,%r15                (in case of S1C33401)
     ```
     Do not alter the `%r15` register after that.

8.9 Error/Warning Messages

The following shows the principal error and warning messages output by the assembler **as**:

Table 8.9.1 Error messages

Error message	Description
`Error: Unrecognized opcode: '`*XXXXX*`'`	The operation code *XXXXX* is undefined.
`Error: `*XXXXXX*`: invalid operand`	A format error of the operand.
`Error: `*XXXXXX*`: invalid register name`	The specified register cannot be used.
`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 `malloc()` 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 `-mc33_ext` option is specified. Or the specified dump file does not exist.
`Error: The format of the dump file is invalid.`	The contents in the dump file specified with the `-mc33_ext` option are invalid.
`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 `-mc33_ext` option has exceeded 2,047 characters.
`Error: Cannot find the all objects' dump file.`	An all-object dump file name is not specified even though the `-mc33_ext` option is specified. Or the specified file does not exist.
`Error: The format of the all objects' dump file is invalid.`	The contents in the all-object dump file specified with the `-mc33_ext` option are invalid.

Table 8.9.2 Warning messages

Warning message	Description
`Warning: operand out of range (`*XXXXXX*`: `*XXX*` not between `*AAA*` and `*BBB*`)`	The value specified in the operand is out of the effective range. It has been corrected within the range (*AAA*–*BBB*).
`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. `.long`, `.int`) exceeds the maximum size. It has been corrected to *AAA*-byte size. (e.g. `0x100000012` → `0x12`)
`Warning: Value `*XXXX*` truncated to `*AAA*	The constant declared exceeds the maximum value *AAA*. It has been corrected to *AAA*. (e.g. `.byte 0x100000012` → `.byte 0xff`)

8.10 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.
- Some assembler extended instructions have a "`symbol+imm26`" operand, note, however, that the valid range that can be specified is as follows:
  0 ≤ (`symbol+imm26-__dp`) < 0x04000000

  Furthermore, in the extended branch instructions that have a "`label+imm32`" operand, the short call/jump instructions (`scall`, `sjp`, etc.) actually allow branching within a ±2MB range, and the long call/jump instructions (`xcall`, `xjp`, etc.) allow branching within a ±2GB range. If the specified branch distance exceeds this range, an error will occur in the linkage stage.
- 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, bit 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.
- When executing a build without the `-medda32` option, the settings for data pointer shown below are required. Otherwise, no optimization for extended instructions will be performed.

  1. Specify the data pointer value in the linker script file.
     Write the command shown below in the linker script file.
     ```
     __dp=0xN;
     ```
     This command is required to set the data pointer. In the **IDE**, it can be specified in the [GNU33 Linker Script Settings] page on the project property dialog box.
  2. Set the `__dp` value to the `%r15` and `%dp` registers in the boot routine.
     Substitute the `__dp` symbol value into the `%r15` register.
     In the S1C33401, the same value must be set to the `%dp` register.
     ```
     xld.w   %r15,__dp
     ld.w    %dp,%r15                 (in case of S1C33401)
     ```
     Do not alter the `%r15` register after that.

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

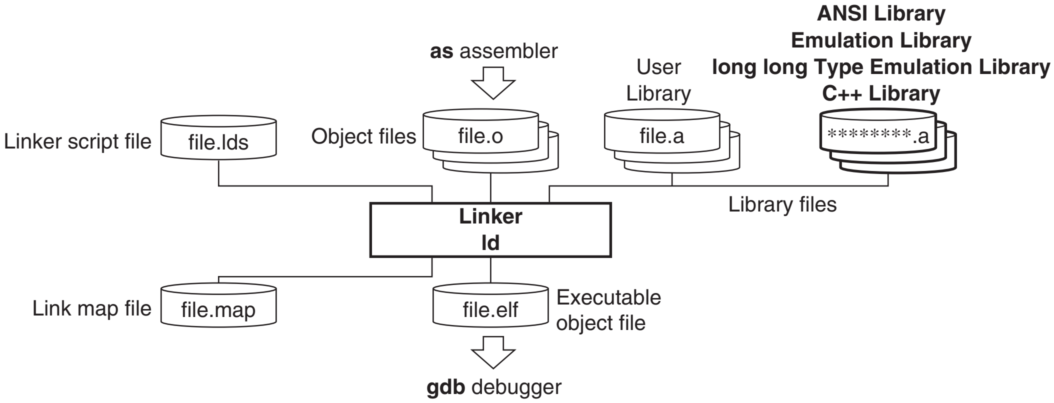

ANSI Library
Emulation Library
long long Type Emulation Library
C++ Library
as assembler
User Library
Linker script file
file.lds
Object files
file.o
file.a
********.a
Library files
Linker
ld
Link map file
file.map
file.elf
Executable object file
gdb debugger

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, `long long` type emulation library files, C++ library files and user library files.

Note: The object and library files to be input to the linker must be compiled with the same target CPU (C33 STD, C33 PE or S1C33401) specified. Object files for different target CPU cannot be linked.

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 ore more library file names.

Example: `ld -o sample.elf boot.o sample.o ..\lib\libc.a ..\lib\libgcc.a -defsym _dp=0`

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 9.4.2) 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\libc.a`

9.4 Linkage

9.4.1 Specifying Data Area Pointer

When a data area is used in the program, the data area pointer (data area start address) must be specified in the linkage stage. It can be specified in the linker script file or using the `-defsym` option in the command line for invoking the **ld** linker.

Example: When specifying the data area pointer as follows:

__dp (default data area pointer): `0xc00000`

Specifying with the **`-defsym`** option

```
ld -defsym __dp=0xc00000 -o sample.elf sample.o
```

Specifying in the linker script file (**`sample.lds`**)

The following should be described in the linker script file:

```
SECTIONS
{
      __dp   = 0xc00000;
                :
```

Use the `-T` option to specify the linker script file in the command line as follows:

```
ld -o sample.elf sample.o -T sample.lds
```

9.4.2 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. If the default data area pointer (__dp) is specified using the `-defsym` option, the __dp setting in the default script are ignored.

```
OUTPUT_FORMAT("elf32-c33", "elf32-c33",
              "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);
SECTIONS
{
  __dp = 0x0;

  . = 0x0;
  .bss    : { *(.bss)    }
  .data   : { *(.data)   }

  . = 0xc00000;
  .text   : { *(.text)   }
  .rodata : { *(.rodata)}
}
```

In this script, data will be located from address 0 in order of `.bss` and `.data` sections, the program codes and constant data will be located from address 0xc00000. Figure 9.4.2.1 shows the memory map after linkage.

Note that this script cannot be used for applications that use the `.data` section in which initialized data is stored because the load memory address (LMA) different from the virtual memory address (VMA) cannot be specified.

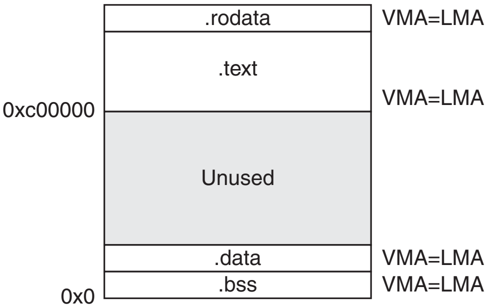


Figure 9.4.2.1 Memory map configured by default script

Linker script generated by the IDE default setting

The default **IDE** settings will create a linker script file similar to the one shown below.

Example: When `boot.o` and `main.o` are linked

```
/* Linker Script file generated by Gnu33 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c33", "elf32-c33", "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);

SECTIONS
{
  /* data pointer symbols */
  __dp = 0x00000000;

  /* location counter */
  . = 0x0;

  /* section information */
  .bss 0x00000000 :
    {
     __START_bss = . ;
     boot.o(.bss)
     main.o(.bss)
     C:/EPSON/gnu33/lib/std/libstdio.a(.bss)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.bss)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.bss)
     C:/EPSON/gnu33/lib/std/libc.a(.bss)
     C:/EPSON/gnu33/lib/std/libgcc.a(.bss)
    }
    __END_bss = . ;

  .comm __END_bss :
    {
     __START_comm = . ;
     boot.o(.comm)
     main.o(.comm)
     C:/EPSON/gnu33/lib/std/libstdio.a(.comm)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.comm)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.comm)
     C:/EPSON/gnu33/lib/std/libc.a(.comm)
     C:/EPSON/gnu33/lib/std/libgcc.a(.comm)
    }
    __END_comm = . ;

  .data __END_comm : AT( __END_gcc_except_table )
    {
     __START_data = . ;
     boot.o(.data)
     main.o(.data)
     C:/EPSON/gnu33/lib/std/libstdio.a(.data)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.data)
     C:/EPSON/gnu33/lib/std/libgcc2.a(.data)
     C:/EPSON/gnu33/lib/std/libc.a(.data)
     C:/EPSON/gnu33/lib/std/libgcc.a(.data)
    }
    __END_data = . ;

  .vector 0x00C00000 :
    {
     __START_vector = . ;
     boot.o(.rodata)
    }
    __END_vector = . ;

  .text __END_vector :
    {
     __START_text = . ;
     boot.o(.text)
     main.o(.text)
     C:/EPSON/gnu33/lib/std/libstdio.a(.text)
     C:/EPSON/gnu33/lib/std/libstdc++.a(.text)
```

```
      C:/EPSON/gnu33/lib/std/libgcc2.a(.text)
      C:/EPSON/gnu33/lib/std/libc.a(.text)
      C:/EPSON/gnu33/lib/std/libgcc.a(.text)
     }
     __END_text = . ;

   .rodata __END_text :
     {
      __START_rodata = . ;
      main.o(.rodata)
      C:/EPSON/gnu33/lib/std/libstdio.a(.rodata)
      C:/EPSON/gnu33/lib/std/libstdc++.a(.rodata)
      C:/EPSON/gnu33/lib/std/libgcc2.a(.rodata)
      C:/EPSON/gnu33/lib/std/libc.a(.rodata)
      C:/EPSON/gnu33/lib/std/libgcc.a(.rodata)
     }
     __END_rodata = . ;

   .ctors __END_rodata :
     {
      . = ALIGN(4);
      __START_ctors = . ;
      boot.o(.ctors)
      main.o(.ctors)
      C:/EPSON/gnu33/lib/std/libstdio.a(.ctors)
      C:/EPSON/gnu33/lib/std/libstdc++.a(.ctors)
      C:/EPSON/gnu33/lib/std/libgcc2.a(.ctors)
      C:/EPSON/gnu33/lib/std/libc.a(.ctors)
      C:/EPSON/gnu33/lib/std/libgcc.a(.ctors)
     }
     __END_ctors = . ;

   .dtors __END_ctors :
     {
      . = ALIGN(4);
      __START_dtors = . ;
      boot.o(.dtors)
      main.o(.dtors)
      C:/EPSON/gnu33/lib/std/libstdio.a(.dtors)
      C:/EPSON/gnu33/lib/std/libstdc++.a(.dtors)
      C:/EPSON/gnu33/lib/std/libgcc2.a(.dtors)
      C:/EPSON/gnu33/lib/std/libc.a(.dtors)
      C:/EPSON/gnu33/lib/std/libgcc.a(.dtors)
     }
     __END_dtors = . ;

   .gcc_except_table __END_dtors :
     {
      __START_gcc_except_table = . ;
      boot.o(.gcc_except_table)
      main.o(.gcc_except_table)
      C:/EPSON/gnu33/lib/std/libstdio.a(.gcc_except_table)
      C:/EPSON/gnu33/lib/std/libstdc++.a(.gcc_except_table)
      C:/EPSON/gnu33/lib/std/libgcc2.a(.gcc_except_table)
      C:/EPSON/gnu33/lib/std/libc.a(.gcc_except_table)
      C:/EPSON/gnu33/lib/std/libgcc.a(.gcc_except_table)
     }
     __END_gcc_except_table = . ;

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

Correct this default script according to the user application using the linker script file editor implemented in the **IDE**. See Section 5.7.7, "Editing a Linker Script", for how to edit linker script files using the **IDE**.

9.4.3 Examples of Linkage

Example 1) when the default data area is used

```
OUTPUT_FORMAT("elf32-c33", "elf32-c33",
              "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);
SECTIONS
{
  /* data pointer symbol By GWB33 */
  __dp = 0x00000000;                          ...1

  . = 0x0;

  .bss :                                      ...2
    {
     __START_bss = . ;
     *(.bss) ;
    }
    __END_bss = . ;

  .data __END_bss : AT( __END_rodata )        ...3
    {
     __START_data = . ;
     *(.data) ;
    }
    __END_data = . ;
    __START_data_lma = LOADADDR( .data );

  .text 0x00c00000 :                          ...4
    {
     __START_text = . ;
     *(.text) ;
    }
    __END_text = . ;

  .rodata __END_text :                        ...5
    {
     __START_rodata = . ;
     *(.rodata) ;
    }
    __END_rodata = . ;

}
```

1. The default data area pointer (`__dp`) is set to 0x0.
2. All `.bss` sections in the input files are located beginning with address 0x0 as a `.bss` section. These sections do not have an actual code, so it is not necessary to specify the load memory address.
3. The memory space immediately following the `.bss` section are allocated to the `.data` sections in the input files. The initial data (actual code) is located following the `.rodata` section. The LMA is specified by the `AT` statement. `__END_rodata` is the symbol defined in the `.rodata` command and it indicates the location counter value immediately following the `.rodata` section.

`__START_data_lma` is defined to refer the load memory address (LMA) of the `.data` section. `__START_data` and `__END_data` are defined to refer the start and end virtual memory addresses (VMA) of the `.data` section. They can be used to copy data from LMA to VMA in the initialize routine as follows:

```
      xld.w  %r0,__START_data_lma
      xld.w  %r1,__START_data
      xld.w  %r2,__END_data
loop:
      ld.b   %r3,[%r0]+
      ld.b   [%r1]+,%r3
      cmp    %r1,%r2
      jrne   loop
```

Note: When a symbol defined in the linker script is referred from C/C++ code, an error may result in the linkage process depending on the symbol value.

```
extern char __START_data_lma,__START_data,__END_data;

char *src = &__START_data_lma;       /* Proper value cannot be acquired. */
char *dst = &__START_data;           /* Proper value cannot be acquired. */
```

For example, "`char *src = &__START_data_lma;`" is compiled as shown below.

```
ext  doff_hi(__START_data_lma)      ;13 high-order bits
ext  doff_lo(__START_data_lma)      ;13 low-order bits
ld.b %r4,[%r15]
```

If the `__START_data_lma` symbol value (address in this case) exceeds 0x3ffffff, the address (32-bit value) cannot be acquired due to occurrence of an overflow. This problem is unique to the S5U1C33001C that supports the data area method, and does not occur in other GNU compilers such as the GNU i386 compiler.

4. All `.text` sections in the input files are located beginning with address 0xc00000.
5. All `.rodata` sections in the input files are located immediately following the `.text` section. `__END_rodata` is defined for specifying the LMA of the `.data` section.

Figure 9.4.3.1 shows the memory map configured with this sample script.

```
ld -o sample.elf file1.o file2.o -T default.lds
```

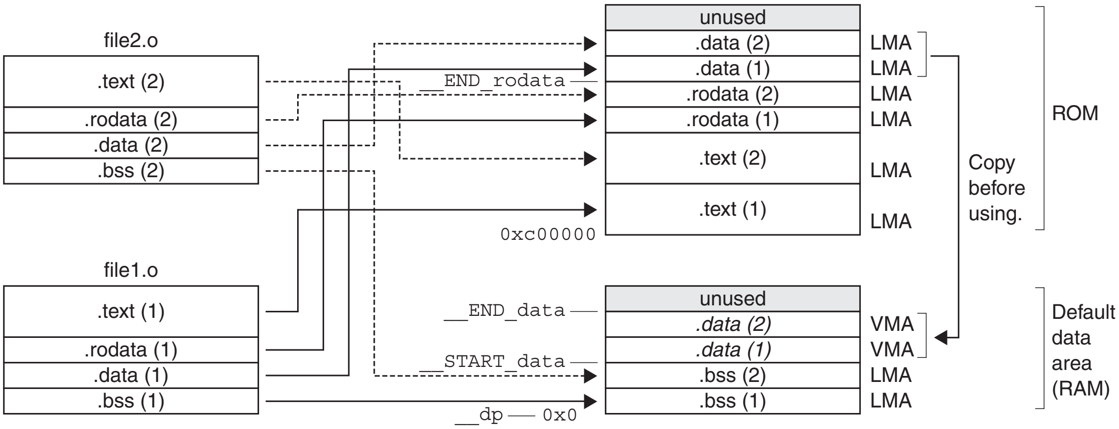

file2.o
.text (2)
.rodata (2)
.data (2)
.bss (2)
file1.o
.text (1)
.rodata (1)
.data (1)
.bss (1)
__END_rodata
0xc00000
__END_data
__START_data
__dp
0x0
unused
.data (2)
.data (1)
.rodata (2)
.rodata (1)
.text (2)
.text (1)
unused
.data (2)
.data (1)
.bss (2)
.bss (1)
LMA
LMA
LMA
LMA
LMA
LMA
VMA
VMA
LMA
LMA
Copy before using.
ROM
Default data area (RAM)

Figure 9.4.3.1 Memory map (Example 1)

Example 2) when virtual and shared sections are used

The following is a sample linker script when virtual and shared sections are used:

```
OUTPUT_FORMAT("elf32-c33", "elf32-c33",
              "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);
SECTIONS
{
   /* data pointer symbol */
   __dp = 0x0;

   /* section information */
   . = 0x0;

   .bss 0x00000000 :
     {
      __START_bss = . ;
      *(.bss) ;
     }
     __END_bss = . ;

   .data __END_bss : AT( __END_rodata )
     {
      __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 );

   .text 0x00600000 :
     {
      __START_text = . ;
      *(.text) ;
     }
     __END_text = . ;

   .rodata __END_text :
     {
      __START_rodata = . ;
      *(.rodata) ;
     }
     __END_rodata = . ;
}
```

The section map is shown in Figure 9.4.3.2.

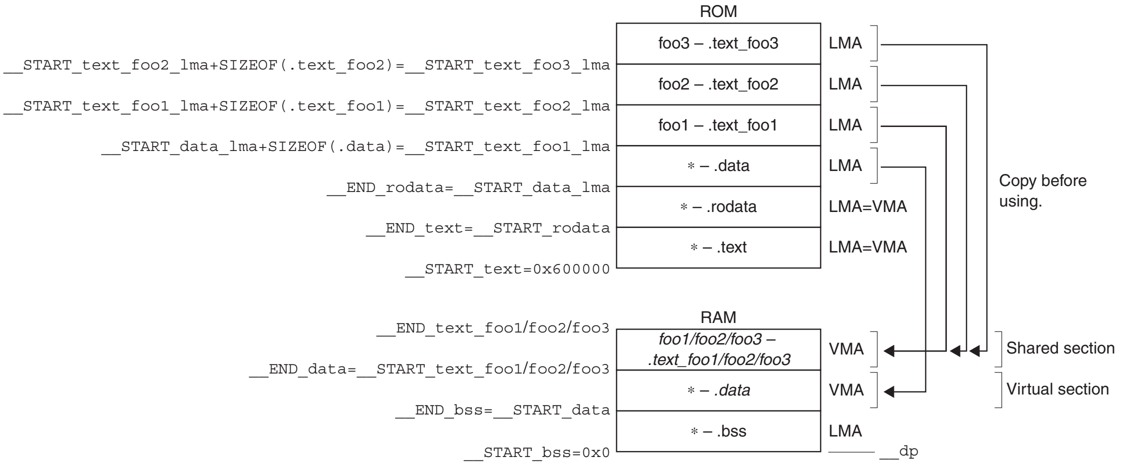

ROM
__START_text_foo2_lma+SIZEOF(.text_foo2)=__START_text_foo3_lma
__START_text_foo1_lma+SIZEOF(.text_foo1)=__START_text_foo2_lma
__START_data_lma+SIZEOF(.data)=__START_text_foo1_lma
__END_rodata=__START_data_lma
__END_text=__START_rodata
__START_text=0x600000
foo3 – .text_foo3
foo2 – .text_foo2
foo1 – .text_foo1
∗ – .data
∗ – .rodata
∗ – .text
LMA
LMA
LMA
LMA
LMA=VMA
LMA=VMA
Copy before using.
RAM
__END_text_foo1/foo2/foo3
__END_data=__START_text_foo1/foo2/foo3
__END_bss=__START_data
__START_bss=0x0
foo1/foo2/foo3 – .text_foo1/foo2/foo3
∗ – .data
∗ – .bss
VMA
VMA
LMA
__dp
Shared section
Virtual section

Figure 9.4.3.2 Memory map (Example 2)

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. `.text` sections in other files than these three files are located in the `.text` section beginning with 0x600000 and are executed at the stored address in the ROM.

9.5 Error/Warning Messages

Error and warning messages are displayed/output through the Standard Output (stdout).

In the **ld** linker, the following error and warning messages are added to the standard error messages of the gnu linker:

Table 9.5.1 Error messages

Error message	Description
Error: Default Data area pointer value is larger than symbol address value.	The `__dp` (default data area pointer) value exceeds the defined symbol address.
Error: The offset value of a symbol is over 64MB. (default data area)	The symbol-offset value is out of the 64MB range from `__dp` (default data area pointer).
Error: The offset value of a symbol is over 512KB. (default data area)	The symbol-offset value is out of the 512KB range from `__dp` (default data area pointer).
Error: The offset value of a symbol is over 64byte. (default data area)	The symbol-offset value is out of the 64-byte range from `__dp` (default data area pointer).
Error: Cannot link [STD|PE|ADV] object <*objectfile*> included from <*archivefile*> with [STD|PE|ADV] object <*first objectfile*>	The object file <*objectfile*> included in the archived file <*archivefile*> cannot be linked with <*first objectfile*>, as the target CPU is different.
Error: Input object file <*objectfile*> [included from <*archivefile*>] is not for C33.	The object file <*objectfile*> included in the archived file <*archivefile*> is not a C33 object file.

Table 9.5.2 Warning message

Warning message	Description
Warning: `__dp` symbol cannot be referred to.	1. The `doff_hi`, `dpoff_h`, `dpoff_m`, `dpoff_l` or `doff_lo` pseudo-operand is used without specifying `__dp`. 2. A `[`*symbol+imm*`]` operand is included in the assembler source without specifying `__dp`. (An instruction with a `[`*symbol+imm*`]` operand is expanded into a code that references the `__dp` value.) `__dp` will be treated as 0x0.

9.6 Precautions

- The object and library files to be input to the linker must be compiled with the same target CPU (C33 STD, C33 PE or S1C33401) specified. Object files for different target CPU cannot be linked.

- 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`)

  ```
        :
  __dp = 0;

  .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) ;
  }
  ```

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 S5U1C33000H, S5U1C33001H, or debug monitor, the debugger incorporates a software simulator function for debugging programs on a personal computer.
- Capable of C/C++ source and assembly source level debugging.
- Supports C/C++ source and assembler level single-stepping functions, in addition to continuous program execution.
- Supports hardware and software PC break functions, and a data break function with memory access conditions specified.
- Can measure program execution time by duration of time or number of cycles.
- Can save traced data.
- Can automatically execute commands using a command file.
- Supports a simulated I/O function that allows input/output evaluation in the debugger.
- Supports the flash writer function of the S5U1C33001H.
- Allows storing and re-registering the symbols registered in the Watch Window.
- Can save/reset breakpoints.

10.2 Input/Output Files

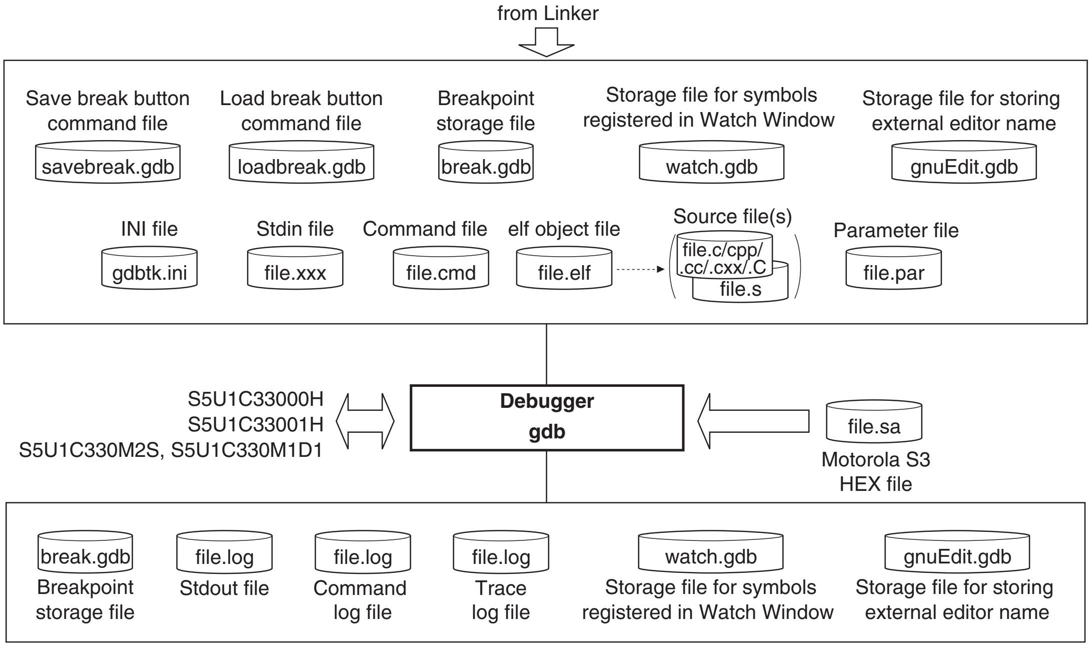

from Linker
Save break button command file
savebreak.gdb
Load break button command file
loadbreak.gdb
Breakpoint storage file
break.gdb
Storage file for symbols registered in Watch Window
watch.gdb
Storage file for storing external editor name
gnuEdit.gdb
INI file
gdbtk.ini
Stdin file
file.xxx
Command file
file.cmd
elf object file
file.elf
Source file(s)
file.c/cpp/.cc/.cxx/.C
file.s
Parameter file
file.par
S5U1C33000H
S5U1C33001H
S5U1C330M2S, S5U1C330M1D1
Debugger
gdb
file.sa
Motorola S3 HEX file
break.gdb
Breakpoint storage file
file.log
Stdout file
file.log
Command log file
file.log
Trace log file
watch.gdb
Storage file for symbols registered in Watch Window
gnuEdit.gdb
Storage file for storing external editor name

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 memory map information for the debugger. The [Project > Properties > GNU33 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 `\gnu33\utility\share` directory to the **gdb.exe** directory (`\gnu33`) 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`/`.cpp`/`.cc`/`.cxx`/`.C` (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 `c33 fwlp` or `c33 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 -I elf32-little -O srec --srec-forceS3` <*filename*>`.elf` <*filename*>`.sa` Example: `objcopy -I elf32-little -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 `c33 stdin` command is used to specify the file name and other information.

Save breakpoints / watch button command file

File format: Text file
File name: `savebreak.gdb`
Description: Same as the command file.

Load breakpoints / watch button command file

File format: Text file
File name: `loadbreak.gdb`
Description: Same as the command file.

Breakpoint storage file

File format: Text file
File name: `break.gdb`
Description: Breakpoint information

Storage file for symbols registered in Watch Window

File format: Text file
File name: `watch.gdb`
Description: Symbol names

Storage file for storing external editor name

File format: Text file
File name: `gnuEdit.gdb`
Description: This file is a command file for storing the external editor name and parameters.

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 `c33 tf`, `c33 otf` or `c33 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 `c33 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 `c33 stdout` command is used to specify a file name and other information.

Breakpoint storage file

File format: Text file
File name: `break.gdb`
Description: This file contains breakpoint data.

Storage file for symbols registered in Watch Window

File format: Text file
File name: `watch.gdb`
Description: Symbol names

Storage file for storing external editor name

File format: Text file
File name: `gnuEdit.gdb`
Description: This file is a command file for storing the external editor name and parameters.

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.

Operation on IDE

After setting the necessary startup commands in the [Project > Properties > GNU33 GDB Commands] dialog box displayed, then Properties, select [GDB33 Launch for *<project name>*] in the [External Tools] dialog box and click the [Run] button. For more information, refer to Section 5.8, "Starting the Debugger."

Precautions

- Except when a second S5UIC33001H is connected using the "`target icd usb2`" command, avoid launching a second instance of the debugger. If invoked twice, the debugger may not operate normally.
- About the SHARE file for GUI debugging use
  A group of files located in the `\gnu33\utility\share` directory are the files written in TCL/TK language, which are used in GUI mode of the debugger. As these are text files, they can be altered by an editor or the like. However, care must be taken, as the debugger may become inoperable depending on the altered content. If it is necessary to alter the content, be sure to gain a full understanding of the detailed specifications before actually making a change.
  Some of these files have been customized from insight gdb for use with the S5U1C33001C manufactured by Seiko Epson. The `\gnu33\utility\share\modify` directory includes a file indicating the customized file names and the contents of modification.
- Do not remove the icon files, such as `\gnu33\utility\share\gdbtcl\images2\userdefine.gif` for the [User Command] button, as the debugger cannot start up normally.

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.

`--c33_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.

`--c33_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.

`--double_starting`

Function: **Enables double starting**

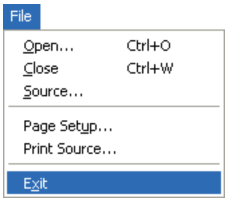
Explanation: This option enables launching a second instance of the debugger that is disabled by default. To debug programs using two S5U1C33001H, specify this option when launching the debugger. Otherwise, avoid launching a second instance of the debugger.

When entering options on the command line, insert one or more spaces to delimit each option.

Example: `c:\EPSON\gnu33\gdb -x sample.cmd --cd=/cygdrive/d/test/sample`

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
```

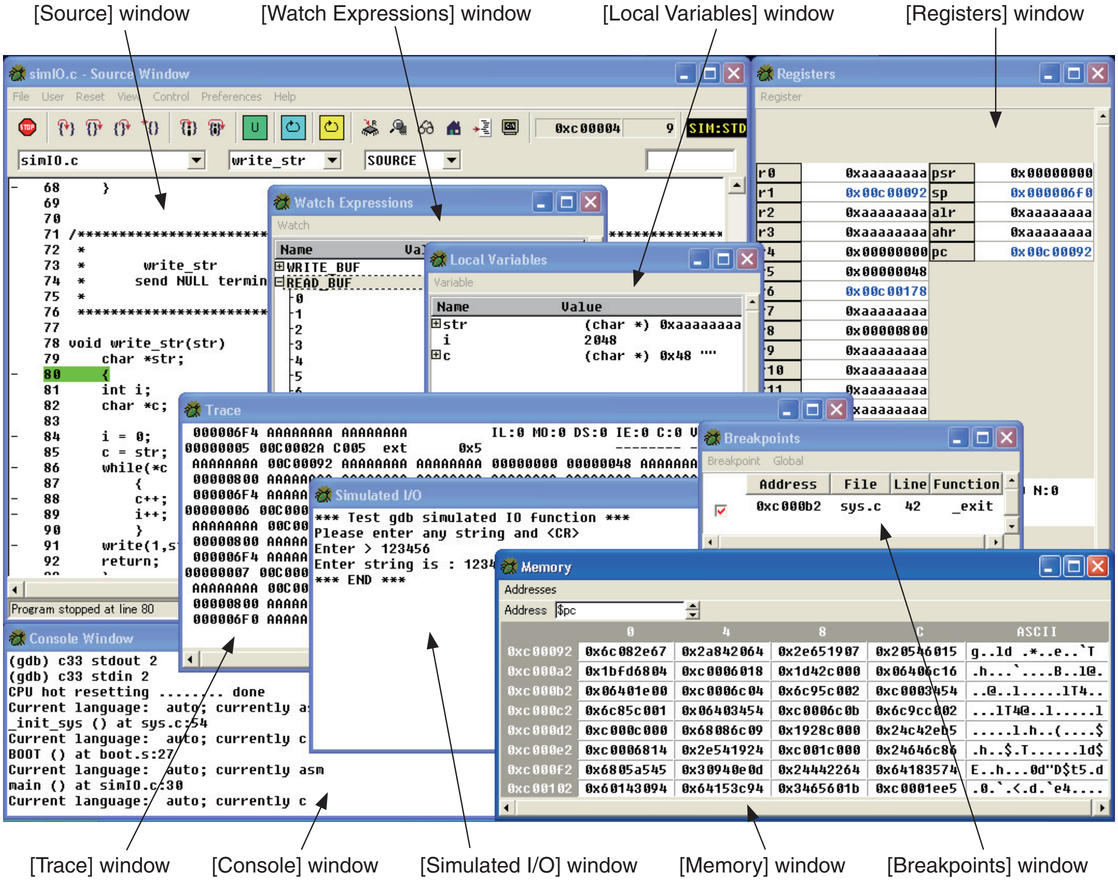
Note: When using the S5U1C33001H to debug a program, always be sure to close the debugger before turning off power to the S5U1C33001H. Should you turn off power to the S5U1C33001H while running the debugger, you will be unable to reconnect it. In such case, you need to restart your computer.

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.


[Source] window
[Watch Expressions] window
[Local Variables] window
[Registers] window
simIO.c - Source Window
File User Reset View Control Preferences Help
0xc00004
9
SIM:STD
simIO.c
write_str
SOURCE
Watch Expressions
Watch
Local Variables
Variable
Registers
Register
Trace
Breakpoints
Breakpoint Global
Simulated I/O
Memory
Addresses
Address $pc
Console Window
[Trace] window
[Console] window
[Simulated I/O] window
[Memory] window
[Breakpoints] window

```
-    68      }
     69
     70
     71 /************************
     72  *
     73  *        write_str
     74  *       send NULL termin
     75  *
     76  ************************
     77
     78 void write_str(str)
     79      char *str;
-    80      {
     81      int i;
     82      char *c;
     83
-    84      i = 0;
     85      c = str;
-    86      while(*c
     87          {
-    88          c++;
-    89          i++;
     90          }
-    91      write(1,s
     92      return;
```

Program stopped at line 80

```
Name                 Val
⊞WRITE_BUF
⊟READ_BUF
 ├0
 ├1
 ├2
 ├3
 ├4
 ├5
```

```
Name           Value
⊞str           (char *) 0xaaaaaaaa
 i             2048
⊞c             (char *) 0x48 ""
```

```
r0    0xaaaaaaaa  psr   0x00000000
r1    0x00c00092  sp    0x000006f0
r2    0xaaaaaaaa  alr   0xaaaaaaaa
r3    0xaaaaaaaa  ahr   0xaaaaaaaa
r4    0x00000000  pc    0x00c00092
r5    0x00000048
r6    0x00c00178
r7    0xaaaaaaaa
r8    0x00000800
r9    0xaaaaaaaa
r10   0xaaaaaaaa
r11   0xaaaaaaaa
      xaaaaaaaa
```

```
 000006F4 AAAAAAAA AAAAAAAA          IL:0 MO:0 DS:0 IE:0 C:0 V
00000005 00C0002A C005  ext      0x5                  --------- -
 AAAAAAAA 00C00092 AAAAAAAA AAAAAAAA 00000000 00000048 AAAAAAA
 00000800 AAAAA
 000006F4 AAAA
00000006 00C000
 AAAAAAAA 00C00
 00000800 AAAAA
 000006F4 AAAAA
00000007 00C000
 AAAAAAAA 00C00
 00000800 AAAAA
 000006F0 AAAAA
```

```
*** Test gdb simulated IO function ***
Please enter any string and <CR>
Enter > 123456
Enter string is : 1234
*** END ***
```

```
   Address    File    Line  Function
☑  0xc000b2   sys.c   42    _exit
```

```
(gdb) c33 stdout 2
(gdb) c33 stdin 2
CPU hot resetting ........ done
Current language:  auto; currently a
_init_sys () at sys.c:54
Current language:  auto; currently c
BOOT () at boot.s:27
Current language:  auto; currently asm
main () at simIO.c:30
Current language:  auto; currently c
```

```
            0           4           8           C           ASCII
0xc00092    0x6c082e67  0x2a842064  0x2e651907  0x20546015  g..ld .*..e..`T
0xc000a2    0x1bfd6804  0xc0006018  0x1d42c000  0x06406c16  .h...`....B..l@.
0xc000b2    0x06401e00  0xc0006c04  0x6c95c002  0xc0003454  ..@..l.....lT4..
0xc000c2    0x6c85c001  0x06403454  0xc0006c0b  0x6c9cc002  ...lT4@..l.....l
0xc000d2    0xc000c000  0x68086c09  0x1928c000  0x24c42eb5  .....l.h..(....$
0xc000e2    0xc0006814  0x2e541924  0xc001c000  0x24646c86  .h..$.T......ld$
0xc000f2    0x6805a545  0x30940e0d  0x24442264  0x64183574  E..h...0d"D$t5.d
0xc00102    0x60143094  0x64153c94  0x3465601b  0xc0001ee5  .0.`.<.d.`e4....
```

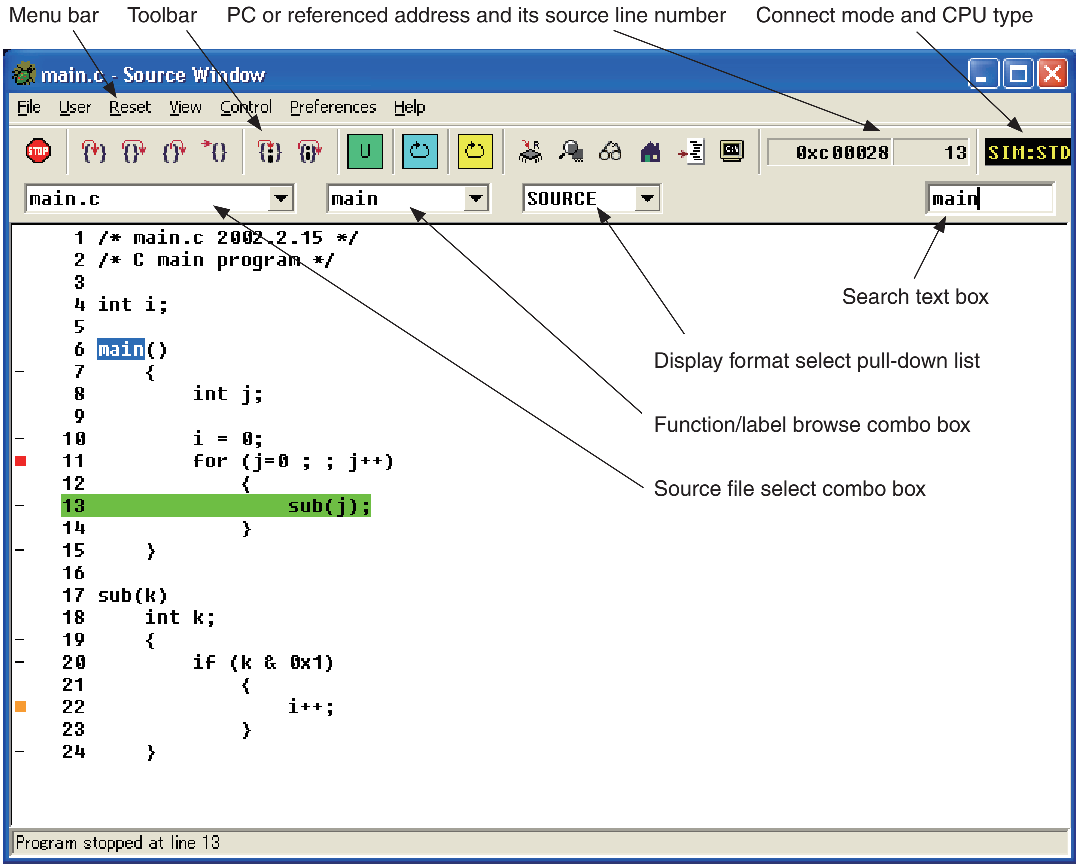
At debugger startup, the `gdbtk.ini` file is loaded in the debugger, with the [Source], [Console], [Registers] and [Memory] windows opening by default. If the `gdbtk.ini` file does not exist, the debugger creates a `gdbtk.ini` file with the default settings at startup, and opens these four windows. To 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 `\gnu33\utility\share` directory to the **gdb.exe** directory (`\gnu33`) 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

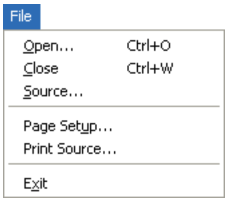

Menu bar
Toolbar
PC or referenced address and its source line number
Connect mode and CPU type
main.c - Source Window
File
User
Reset
View
Control
Preferences
Help
U
0xc00028
13
SIM:STD
main.c
main
SOURCE
main
Search text box
Display format select pull-down list
Function/label browse combo box
Source file select combo box

```
       1 /* main.c 2002.2.15 */
       2 /* C main program */
       3
       4 int i;
       5
       6 main()
-      7     {
       8         int j;
       9
-     10         i = 0;
■     11         for (j=0 ; ; j++)
      12             {
-     13                 sub(j);
      14             }
-     15     }
      16
      17 sub(k)
      18     int k;
-     19     {
-     20         if (k & 0x1)
      21             {
■     22                 i++;
      23             }
-     24     }
```

Program stopped at line 13

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.

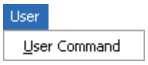
Print Source...

Prints the currently displayed source in exactly the same form as displayed on the screen.

Exit

Quits the debugger.

[User] menu


User
User Command

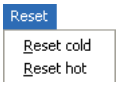
User Command

Executes the command file `\gnu33\userdefine.gdb` that can be edited by the user.

Contents of `userdefine.gdb` at shipment

`#Edit user command`
`#c33 rsth` (No command executed)

[Reset] menu


Reset
Reset cold
Reset hot

Reset cold

Executes the command file `\gnu33\resetcold.gdb` that can be edited by the user.

Contents of `resetcold.gdb` at shipment

`c33 rstc` (Cold-resets the CPU)

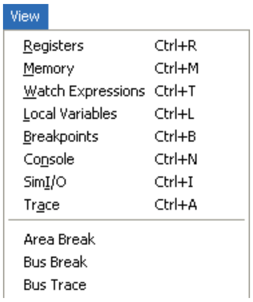
Reset hot

Executes the command file `\gnu33\resethot.gdb` that can be edited by the user.

Contents of `resethot.gdb` at shipment

`c33 rsth` (Hot-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
Area Break
Bus Break
Bus Trace

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.

Area Break

Opens the [On Chip Area Break] dialog box used to set the on-chip area break.

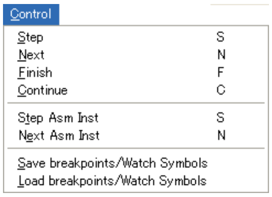
Bus Break

Opens the [On Chip Bus Break] dialog box used to set the on-chip bus break.

Bus Trace

Opens the [On Chip Bus Trace] dialog box used to set the on-chip bus trace.

[Control] menu



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.

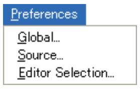
Save breakpoints / Watch Symbols

The gdb debugger stores the breakpoint status and symbol names registered in the Watch Window.

Load breakpoints / Watch Symbols

Sets the breakpoints and the symbol names in the Watch Window as stored in the gdb debugger.

[Preferences] menu



Global...

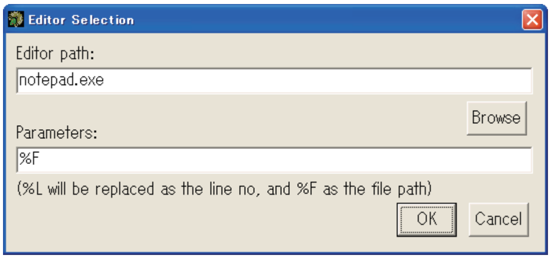
Displays the [Global Preferences] dialog box used to select the fonts. For details about the content set in this dialog box, see "Changing the basic settings" later in this section.

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.

Editor Selection

Displays the [Editor Selection] dialog box used for setting the external editor to use.



Editor path: notepad.exe

Parameters: %F

(%L will be replaced as the line no, and %F as the file path)

[Editor path:]

Display the path to the external editor selected via the [Browse] button or use the keyboard to enter the path to the external editor. Up to 255 characters can be entered.

If the [OK] button is pressed with the [Editor path] column left blank, the following error message appears:

`"Please input 'Editor path'."`

If the path selected after the [Browse] button is pressed exceeds 255 characters in length, the following error message appears and the [Editor path] column becomes blank:

`"'Editor path' exceeds 255 characters."`

The default setting is `"notepad.exe"`.

[Parameters:]

Enter the character string of the argument required by the selected editor, if any. Up to 255 characters can be entered. If there is no argument, it can be left blank.

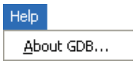
The reserved argument codes are as follows:

%F: Replaced with the file name during editor startup.

%L: Replaced with the line number during startup.

The "%F" parameter is set as the default.

[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. This button is not effective in debug monitor mode.

[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 `\gnu33\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 × 24-dot button image.
`\gnu33\utility\share\gdbtcl\images2\userdefine.gif`

[Reset cold] button

Executes the command file `\gnu33\resetcold.gdb` that can be edited by the user.

[Reset hot] button

Executes the command file `\gnu33\resethot.gdb` that can be edited by the user.

[Save Break / Watch] button

Executes a command file `\gnu33\savebreak.gdb` that can be edited by the user.

[Load break / Watch] button

Executes a command file `\gnu33\loadbreak.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

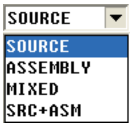
Opens the [Console] window. If already open, the window moves to the front of the screen.

Content displayed in window

Program display

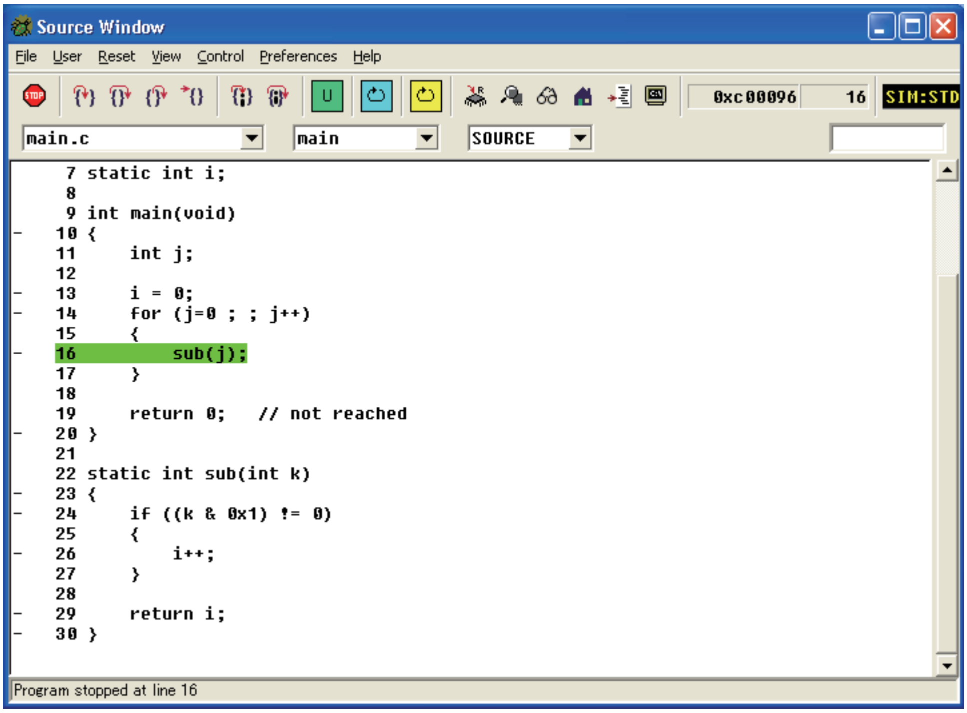
The [Source] window displays the contents of the loaded program. Only one source file at a time can be displayed in this window. If necessary, multiple instances of the [Source] window can be opened, with separate source files displayed in each.

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.

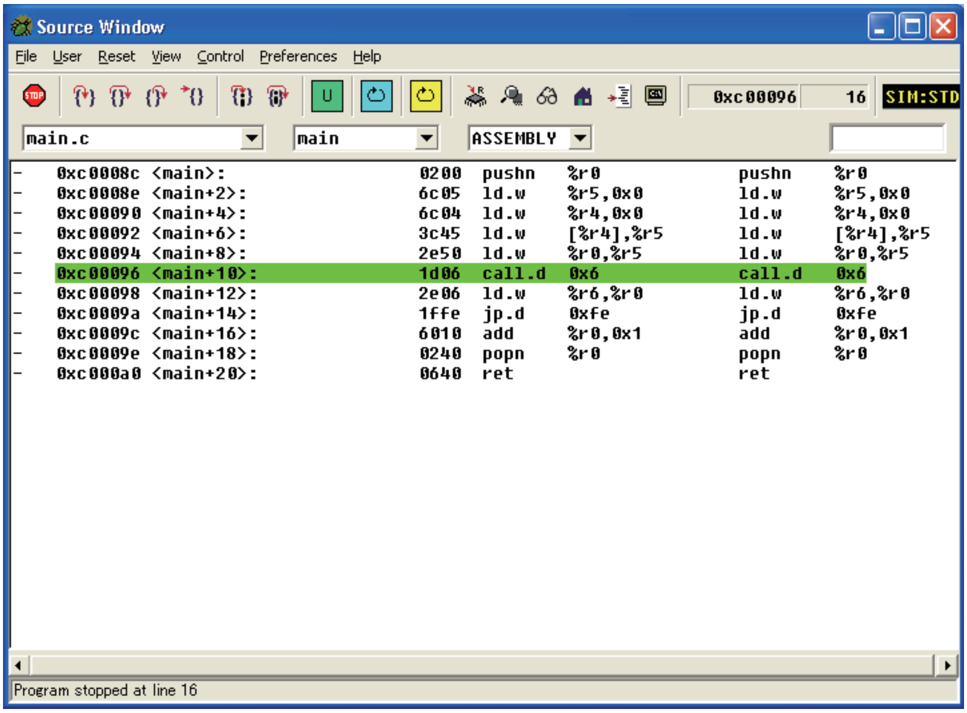

Source Window
File User Reset View Control Preferences Help
STOP
U
0xc00096
16
SIM:STD
main.c
main
SOURCE

```
     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     }
    28
-   29     return i;
-   30 }
```

```
Program stopped at line 16
```

Assembler display (with [ASSEMBLY] selected from pull-down list)

The window shows addresses, labels, and basic and extended assembler instructions.


Source Window
File User Reset View Control Preferences Help
STOP
U
0xc00096
16
SIM:STD
main.c
main
ASSEMBLY

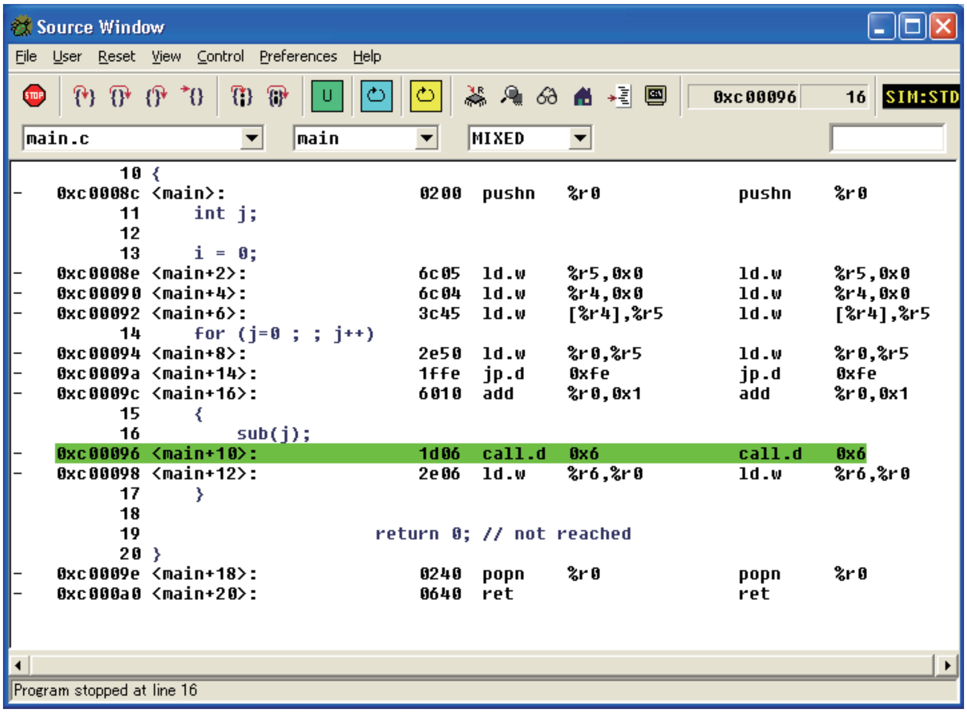
```
-     0xc0008c <main>:            0200  pushn   %r0            pushn    %r0
-     0xc0008e <main+2>:          6c05  ld.w    %r5,0x0        ld.w     %r5,0x0
-     0xc00090 <main+4>:          6c04  ld.w    %r4,0x0        ld.w     %r4,0x0
-     0xc00092 <main+6>:          3c45  ld.w    [%r4],%r5      ld.w     [%r4],%r5
-     0xc00094 <main+8>:          2e50  ld.w    %r0,%r5        ld.w     %r0,%r5
-     0xc00096 <main+10>:         1d06  call.d  0x6            call.d   0x6
-     0xc00098 <main+12>:         2e06  ld.w    %r6,%r0        ld.w     %r6,%r0
-     0xc0009a <main+14>:         1ffe  jp.d    0xfe           jp.d     0xfe
-     0xc0009c <main+16>:         6010  add     %r0,0x1        add      %r0,0x1
-     0xc0009e <main+18>:         0240  popn    %r0            popn     %r0
-     0xc000a0 <main+20>:         0640  ret                    ret
```

```
Program stopped at line 16
```

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.

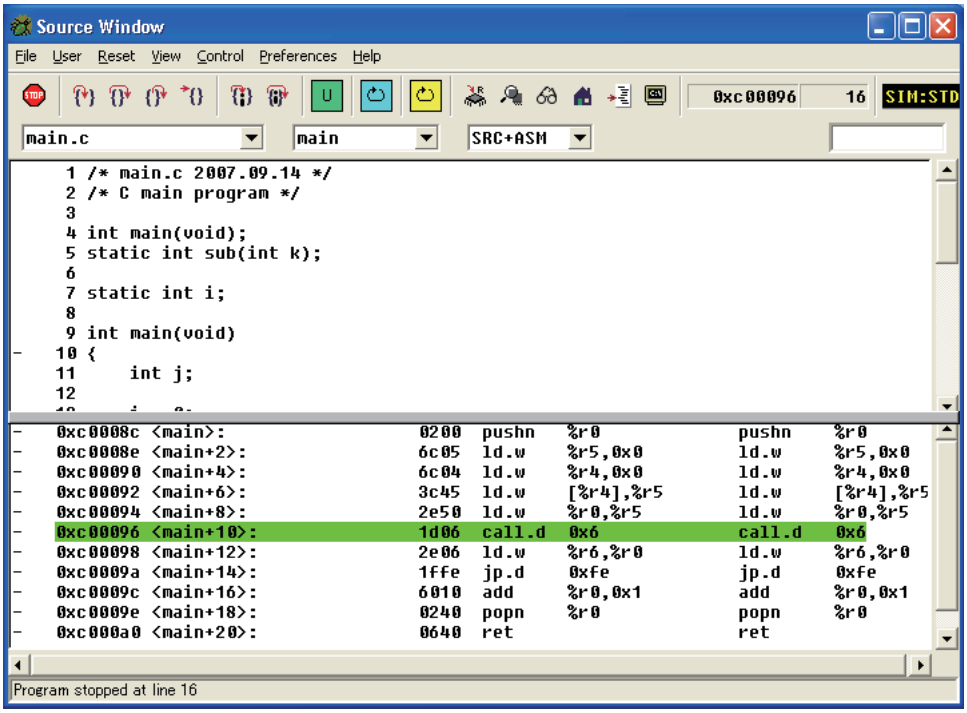

Source Window
File User Reset View Control Preferences Help
STOP
U
0xc00096
16
SIM:STD
main.c
main
MIXED

```
          10 {
-   0xc0008c <main>:                  0200  pushn    %r0           pushn     %r0
          11     int j;
          12
          13     i = 0;
-   0xc0008e <main+2>:                6c05  ld.w     %r5,0x0       ld.w      %r5,0x0
-   0xc00090 <main+4>:                6c04  ld.w     %r4,0x0       ld.w      %r4,0x0
-   0xc00092 <main+6>:                3c45  ld.w     [%r4],%r5     ld.w      [%r4],%r5
          14     for (j=0 ; ; j++)
-   0xc00094 <main+8>:                2e50  ld.w     %r0,%r5       ld.w      %r0,%r5
-   0xc0009a <main+14>:               1ffe  jp.d     0xfe          jp.d      0xfe
-   0xc0009c <main+16>:               6010  add      %r0,0x1       add       %r0,0x1
          15     {
          16         sub(j);
-   0xc00096 <main+10>:               1d06  call.d   0x6           call.d    0x6
-   0xc00098 <main+12>:               2e06  ld.w     %r6,%r0       ld.w      %r6,%r0
          17     }
          18
          19                    return 0; // not reached
          20 }
-   0xc0009e <main+18>:               0240  popn     %r0           popn      %r0
-   0xc000a0 <main+20>:               0640  ret                    ret
```

Program stopped at line 16

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
STOP
U
0xc00096
16
SIM:STD
main.c
main
SRC+ASM

```
       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
```

```
-   0xc0008c <main>:                  0200  pushn    %r0           pushn     %r0
-   0xc0008e <main+2>:                6c05  ld.w     %r5,0x0       ld.w      %r5,0x0
-   0xc00090 <main+4>:                6c04  ld.w     %r4,0x0       ld.w      %r4,0x0
-   0xc00092 <main+6>:                3c45  ld.w     [%r4],%r5     ld.w      [%r4],%r5
-   0xc00094 <main+8>:                2e50  ld.w     %r0,%r5       ld.w      %r0,%r5
-   0xc00096 <main+10>:               1d06  call.d   0x6           call.d    0x6
-   0xc00098 <main+12>:               2e06  ld.w     %r6,%r0       ld.w      %r6,%r0
-   0xc0009a <main+14>:               1ffe  jp.d     0xfe          jp.d      0xfe
-   0xc0009c <main+16>:               6010  add      %r0,0x1       add       %r0,0x1
-   0xc0009e <main+18>:               0240  popn     %r0           popn      %r0
-   0xc000a0 <main+20>:               0640  ret                    ret
```

Program stopped at line 16

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/C++ sources, only the sources in a file compiled after specifying C/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.)

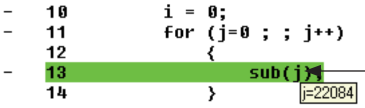
The source lines or addresses marked with 🟥 instead of the bar "–" indicate that PC breakpoints are set there. The color of the mark represents the type or status of the breakpoint.

🟥 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

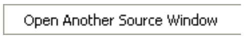
While the program is being executed, the contents displayed in the window are not updated. Instead, the source file or program display positions are updated so that the line at the current PC address is displayed in the window after a break. At the same time, the current PC address and its source line 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.

Launching the external editor

Right-click on a source file in the [Source] window and choose [Open Source in external editor] to open the source file using the editor set in the [Editor Selection] dialog box. (Notepad launches by default.)

The external editor is activated when the display mode is "SOURCE" or "SRC+ASM" and the mouse cursor is positioned on the "SRC" side.

If a source file has been edited and saved using an external editor, executing a program using the .gdb file will display a warning message that the source file has been changed, and the edited source file will be displayed.

Since the edited source file has not been built, it does not match the contents of the currently loaded execution file. In such case, close the debugger and perform a rebuild before resuming debugging.

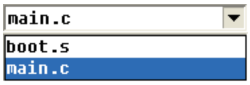
- If no editor is set, the error message "Please set external editor" appears.
- If the specified editor does not exist, the error message "Editor not found" appears.

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].

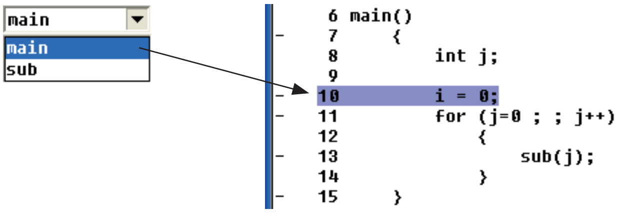
Selecting between source files

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).



```
main()
    {
        int j;

        i = 0;
        for (j=0 ; ; j++)
            {
                sub(j);
            }
    }
```

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].

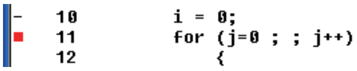
Using the mouse to set and clear breakpoints

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.

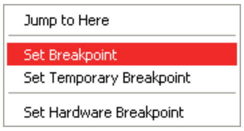
This method can only be used to set normal software PC breakpoints. To clear breakpoints, this method can be used for temporary software PC breakpoints (orange 🟧) and hardware PC breakpoints (blue 🟦), as well as set PC breakpoints that are currently disabled (black ⬛).



```
-    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.

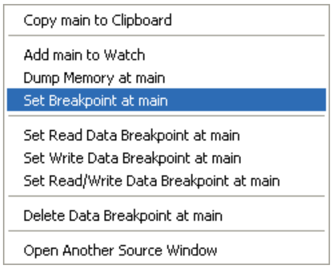
When you choose [Set Temporary Breakpoint], the "–" at the beginning of the line changes to 🟧 (by default, in orange), indicating that a temporary software PC breakpoint has been set at that line, and one that is only effective for one break hit.

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.


Copy main to Clipboard
Add main to Watch
Dump Memory at main
Set Breakpoint at main
Set Read Data Breakpoint at main
Set Write Data Breakpoint at main
Set Read/Write Data Breakpoint at main
Delete Data Breakpoint at main
Open Another Source Window

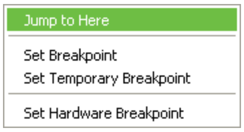
Choose [Set Breakpoint at <symbol>] from this menu to set the source line to be executed after prologue processing in that function to a normal software PC breakpoint. The mark at the beginning of the line changes to 🟥 (by default, in red).

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 up to two locations. 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.


Jump to Here
Set Breakpoint
Set Temporary Breakpoint
Set Hardware Breakpoint

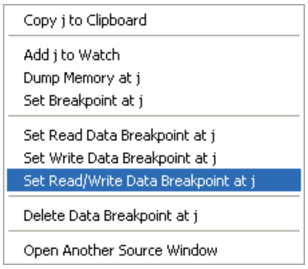
Choose [Jump to Here] from this menu to start running the target program from the current PC address, then stopping it at this line. (This is not effective unless the program processes this line.)

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. However, this function is effective even if 200 software PC breakpoints (max.) have already been set.

Setting and clearing data breakpoints

Data breakpoints can be set and cleared in the window.

First, to use this function, choose display format [SOURCE]. Place the cursor on a global variable name and right-click to display the popup menu shown below.


Copy j to Clipboard
Add j to Watch
Dump Memory at j
Set Breakpoint at j
Set Read Data Breakpoint at j
Set Write Data Breakpoint at j
Set Read/Write Data Breakpoint at j
Delete Data Breakpoint at j
Open Another Source Window

Select [Set Read/Write Data Breakpoint at <*symbol*>] from this menu to set a data breakpoint at which the program breaks upon access to the variable for read/write. If you want the program to break upon access for only read or write, select [Set Read Data Breakpoint at <*symbol*>] or [Set Write Data Breakpoint at <*symbol*>], respectively.

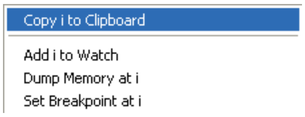
Select [Delete Data Breakpoint at <*symbol*>] to clear the data breakpoint set.

Note: A data break can be set only at one location. The content set is not displayed in the [Source] and other windows. Use the `info breakpoints` command to check. Also note that no data breakpoints can be set for local variables.

Note: If the symbol is of the double, array, or structure type, data break points will not be set.

Copying symbols

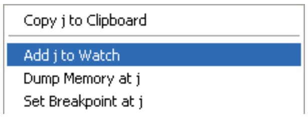
Variable names and function names can be copied to the clipboard using a popup menu. To use this function, change the display format to [SOURCE]. Place the cursor over a variable name, function name, or label, and right-click. This displays the popup menu shown below.


Copy i to Clipboard
Add i to Watch
Dump Memory at i
Set Breakpoint at i

Select [Copy <*symbol*> to Clipboard] from this menu to copy the symbol name to the clipboard.

Registering symbols in the watch list

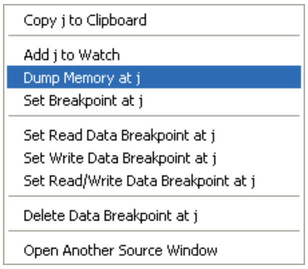
The global symbols you wish to register in the [Watch Expressions] window can be selected from the [Source] window. To use this function, choose display format [SOURCE]. Place the cursor over a global variable name, then right-click to display the popup menu shown below.


Copy j to Clipboard
Add j to Watch
Dump Memory at j
Set Breakpoint at j

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.


Copy j to Clipboard
Add j to Watch
Dump Memory at j
Set Breakpoint at j
Set Read Data Breakpoint at j
Set Write Data Breakpoint at j
Set Read/Write Data Breakpoint at j
Delete Data Breakpoint at j
Open Another Source Window

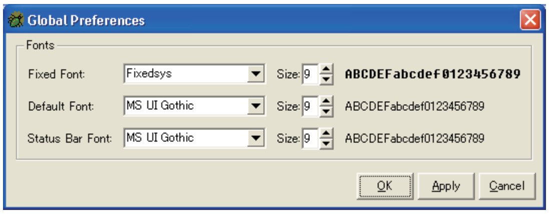
Choose [Dump Memory at <*symbol*>] from this menu to display a new [Memory] window, showing the contents of memory beginning with the selected symbol value.

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.


Global Preferences
OK
Apply
Cancel

Fonts

Fixed Font: Fixedsys Size: 9 ABCDEFabcdef0123456789

Default Font: MS UI Gothic Size: 9 ABCDEFabcdef0123456789

Status Bar Font: MS UI Gothic Size: 9 ABCDEFabcdef0123456789

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

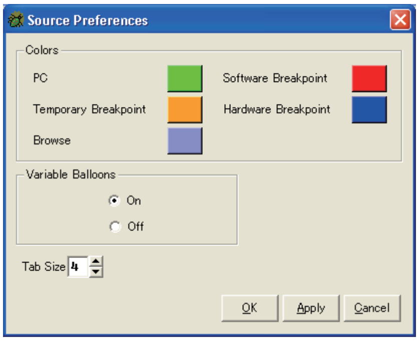
Closes the dialog box without changing the settings you made there. (Note that settings already changed by [Apply] cannot be undone.)

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.



Variable Balloons: ● On ○ Off

Tab Size 4

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:

`c33 rstc` command: [Reset cold] on [Reset] menu, [Reset cold] button *1
`c33 rsth` command: [Reset hot] on [Reset] menu, [Reset hot] button *2
`step` command: [Step] on [Control] menu, [Step] button *4
`stepi` command: [Step Asm Inst] on [Control] menu, [Step Asm Inst] button *4
`next` command: [Next] on [Control] menu, [Next] button *4
`nexti` command: [Next Asm Inst] on [Control] menu, [Next Asm Inst] button *4
`finish` command: [Finish] on [Control] menu, [Finish] button
`continue` command: [Continue] on [Control] menu, [Continue] button *5
User defined command: [User command] on [User] menu, [User Command] button *3

*1 The [Reset cold] command in the [Reset] menu and the [Reset cold] button execute the command file `\gnu33\resetcold.gdb` (the `c33 rstc` command only is included at shipment).

*2 The [Reset hot] command in the [Reset] menu and the [Reset hot] button execute the command file `\gnu33\resethot.gdb` (the `c33 rsth` command only is included at shipment).

*3 The [User command] command in the [User] menu and the [User Command] button execute the command file `\gnu33\userdefine.gdb` (no executable command is included at shipment).

*4 The `step`, `stepi`, `next` and `nexti` commands are executed for only one step from the current PC address.

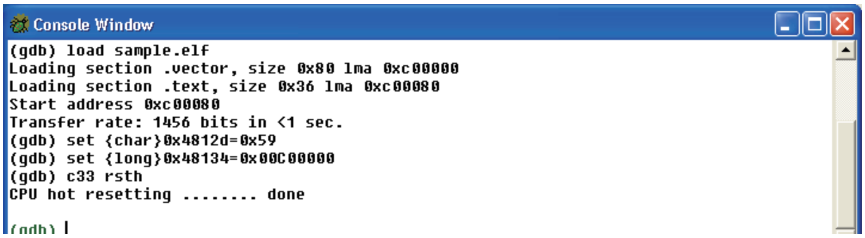
*5 The `continue` command is executed continuously, without specifying the number of invalid breaks.

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.



```
(gdb) load sample.elf
Loading section .vector, size 0x80 lma 0xc00000
Loading section .text, size 0x36 lma 0xc00080
Start address 0xc00080
Transfer rate: 1456 bits in <1 sec.
(gdb) set {char}0x4812d=0x59
(gdb) set {long}0x48134=0x00C00000
(gdb) c33 rsth
CPU hot resetting ........ done

(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 (↑, ↓), 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".

Copy & paste

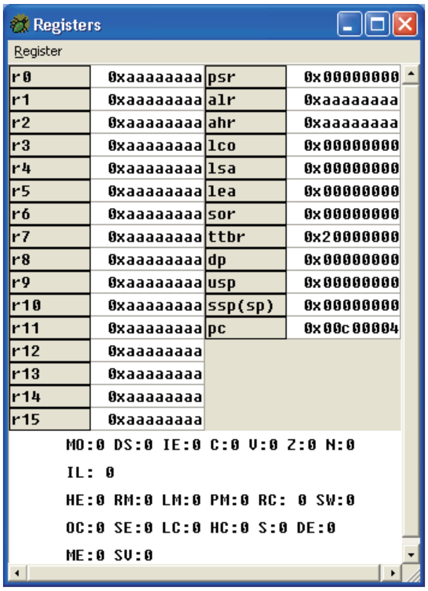
Use keystroke combinations to copy and paste strings in the [Console] window.
Copy: Select a string with the mouse and press the [Ctrl] + [C] key combination.
Paste: Move the mouse to the position at which you want to paste the copied string and press the [Ctrl] + [V] key combination.

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.

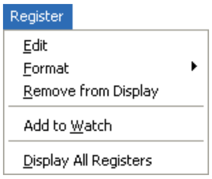


r0	0xaaaaaaaa	psr	0x00000000
r1	0xaaaaaaaa	alr	0xaaaaaaaa
r2	0xaaaaaaaa	ahr	0xaaaaaaaa
r3	0xaaaaaaaa	lco	0x00000000
r4	0xaaaaaaaa	lsa	0x00000000
r5	0xaaaaaaaa	lea	0x00000000
r6	0xaaaaaaaa	sor	0x00000000
r7	0xaaaaaaaa	ttbr	0x20000000
r8	0xaaaaaaaa	dp	0x00000000
r9	0xaaaaaaaa	usp	0x00000000
r10	0xaaaaaaaa	ssp(sp)	0x00000000
r11	0xaaaaaaaa	pc	0x00c00004
r12	0xaaaaaaaa		
r13	0xaaaaaaaa		
r14	0xaaaaaaaa		
r15	0xaaaaaaaa		

```
MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0

IL: 0

HE:0 RM:0 LM:0 PM:0 RC: 0 SW:0

OC:0 SE:0 LC:0 HC:0 S:0 DE:0

ME:0 SV:0
```

Menus

[Register] menu

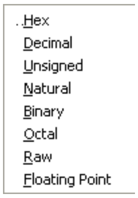


Edit

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:	Hexadecimal (default)
Decimal:	Signed decimal
Unsigned:	Unsigned decimal
Natural:	Signed decimal
Binary:	Binary
Octal:	Octal
Raw:	8-digit hexadecimal
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 display contents depend on the C33 Core being debugged. The register values displayed here are updated after program execution.
C33 STD Core: `r0-r15`, `psr`, `sp`, `alr`, `ahr`, `pc`
C33 ADV Core: `r0-r15`, `psr`, `sp`, `alr`, `ahr`, `pc`, `lco`, `lsa`, `lea`, `sor`, `ttbr`, `dp`, `usp`, `ssp`
C33 PE Core: `r0-r15`, `psr`, `sp`, `alr`, `ahr`, `pc`, `ttbr`

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.

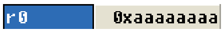
[Registers] button

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: Signed decimal
Binary: Binary
Octal: Octal
Raw: 8-digit hexadecimal
Floating Point: Floating point

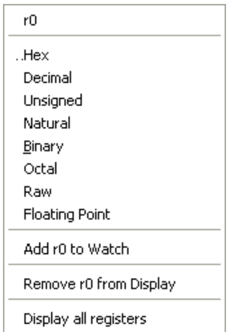
Choose the register whose display format you wish to change by clicking it. (Clicking once more deselects the register.) The name of a selected register is highlighted in blue.



r0 0xaaaaaaaa

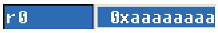
Selected register

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.



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 32 low-order bits are handled as valid, with those after bit 32 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-clinking. 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-clinking. 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

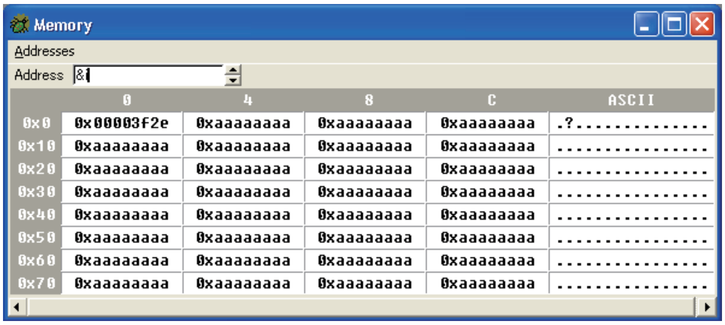
You can add a selected register to the watch list in the [Watch Expressions] window, so that you can monitor it while executing a program.

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

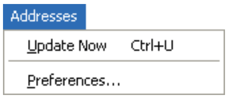
The [Memory] window is used to display and correct the contents of memory.



```
        0           4           8           C           ASCII
0x0     0x00003f2e  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  .?..............
0x10    0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
0x20    0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
0x30    0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
0x40    0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
0x50    0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
0x60    0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
0x70    0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  0xaaaaaaaa  ................
```

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, four words 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.

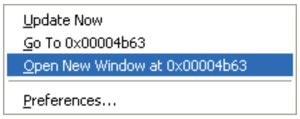
Note that half-word and word data are displayed in little endian format. In simulator mode, the memory content of any area is displayed in endian format (i.e., big or little endian as specified in a parameter file) set for that area.

Operation

Opening and closing the window

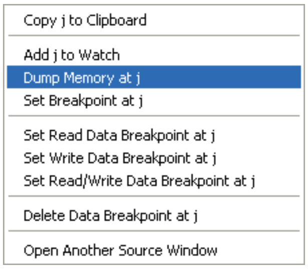
The [Memory] window opens automatically at debugger startup. Clicking the [Close] button closes the window. The window can also be minimized.

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.

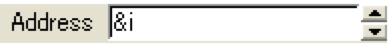
Otherwise, you can right-click a symbol name in the [Source] window and choose [Dump Memory at <*symbol*>] from the ensuing popup menu.


Copy j to Clipboard
Add j to Watch
Dump Memory at j
Set Breakpoint at j
Set Read Data Breakpoint at j
Set Write Data Breakpoint at j
Set Read/Write Data Breakpoint at j
Delete Data Breakpoint at j
Open Another Source Window

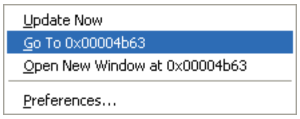
Specifying the display start address

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.



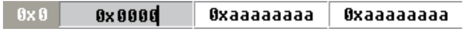
Otherwise, you can select data by clicking it in the [Memory] window, then choosing [Go to <*data*>] from the popup menu that appears by right-clicking. The selected data is then copied to the [Address] text box. Click in the [Address] text box and press the [Enter] key to have the debugger start display from that address.


Update Now
Go To 0x00004b63
Open New Window at 0x00004b63
Preferences...

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

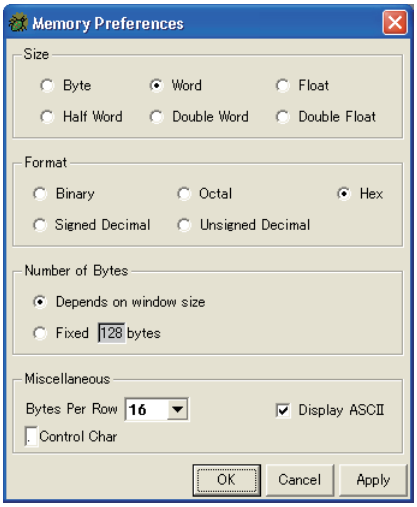
Choose a data cell you wish to change by clicking it in the window. Once a data cell is selected, the cell background is colored and a cursor appears after the value. Thus, you can enter hexadecimal or decimal data (or ASCII characters for entry in the ASCII field).


0x0
0x0000
0xaaaaaaaa
0xaaaaaaaa

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.



Memory Preferences

Size
○ Byte ● Word ○ Float
○ Half Word ○ Double Word ○ Double Float

Format
○ Binary ○ Octal ● Hex
○ Signed Decimal ○ Unsigned Decimal

Number of Bytes
● Depends on window size
○ Fixed 128 bytes

Miscellaneous
Bytes Per Row 16 ☑ Display ASCII
☐ Control Char

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.)

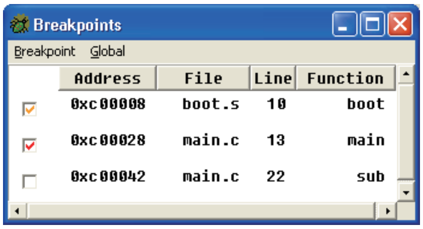
Clicking [Apply] or closing this dialog box updates the contents displayed in the [Source] and other windows.

Limitations

Even when the contents of memory addresses displayed in the [Memory] window are corrected by a command, etc., the contents displayed in the [Memory] window are not changed. To update the displayed contents, choose [Update Now] from the [Addresses] menu or popup menu that appears by right-clicking in the window.

10.4.6 [Breakpoints] Window

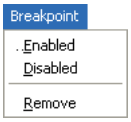
The [Breakpoints] window is used to manage normal software, temporary software and hardware PC breakpoints.



	Address	File	Line	Function
☑	0xc00008	boot.s	10	boot
☑	0xc00028	main.c	13	main
☐	0xc00042	main.c	22	sub

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.

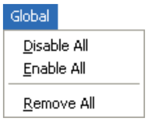
Disabled

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

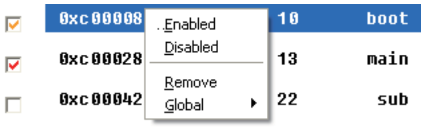
Click the [Close] button to close the window. The window can also be minimized.

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.



☑ 0xc00008 10 boot

☑ 0xc00028 13 main

☐ 0xc00042 22 sub

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 two locations, 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 two 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

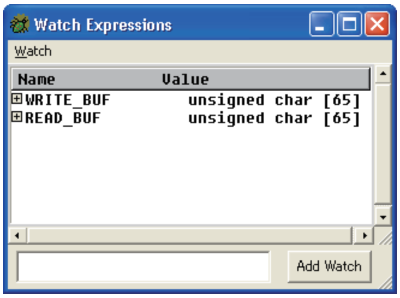
Select a breakpoint you wish to delete by clicking on its line in the breakpoint list and choose [Remove] from the [Breakpoint] menu or popup menu that appears by right-clicking. To delete all PC breakpoints, choose [Remove All] from the [Global] menu or [Global] popup menu that appears by right-clicking.

Precautions

The [Breakpoints] window can only display PC breakpoints. Other breakpoints are not displayed in this window.

10.4.7 [Watch Expressions] Window

The [Watch Expressions] window is used to monitor the values of global symbols and registers.



Name	Value
⊞WRITE_BUF	unsigned char [65]
⊞READ_BUF	unsigned char [65]

Menus

[Watch] menu

Watch
Edit
Format
Remove

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
Decimal
Binary
Octal
Natural

Hex: Hexadecimal
Decimal: Signed decimal
Binary: Binary
Octal: Octal
Natural: Signed decimal (default)

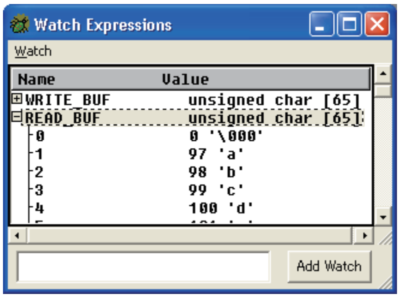
Remove

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.



Name	Value
⊞WRITE_BUF	unsigned char [65]
⊟READ_BUF	unsigned char [65]
0	0 '\000'
1	97 'a'
2	98 'b'
3	99 'c'
4	100 'd'

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.

When binary format is selected, the window displays the symbol values for number of digits equivalent to the symbol size.

Example: `int iSymbol = 0x1a55;`

The size of int type variable is 16 bits, so this symbol value is displayed as "`iSymbol  0001101001010101`".

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.

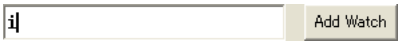
Only one instance of the [Watch Expressions] window can be open at a time. Any attempt to open the [Watch Expressions] window while already open reactivates the [Watch Expressions] window and moves it to the front of the screen.

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

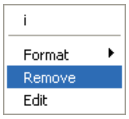
   Select a register you wish to add by clicking it in the [Registers] window, then choose [Add to Watch] from the [Register] menu or the popup menu that appears by right-clicking. (See Section 10.4.4, "[Registers] Window".)

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.

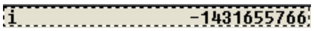


Changing display format

The format in which to display symbol values can be selected from the following:

Hex:	Hexadecimal
Decimal:	Signed decimal
Binary:	Binary
Octal:	Octal
Natural:	Signed decimal (default)

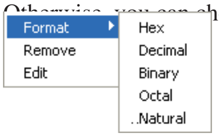
Select a symbol whose display format you wish to change by clicking on its line in the window.



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.

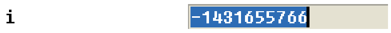


Rewriting symbol values

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.

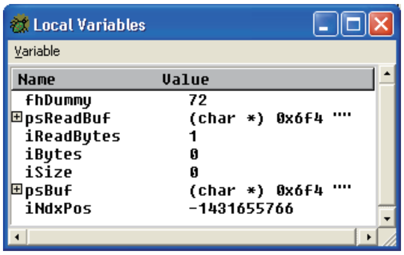
Note that entering a value exceeding the data size of the symbol only applies the low-order effective bits and ignores the bits outside the valid range.

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.

10.4.8 [Local Variables] Window

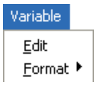
The [Local Variables] window is used to monitor the values of local variables.



```
Name            Value
 fhDummy         72
⊞psReadBuf       (char *) 0x6f4 ""
 iReadBytes      1
 iBytes          0
 iSize           0
⊞psBuf           (char *) 0x6f4 ""
 iNdxPos         -1431655766
```

Menus

[Variable] menu

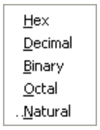


Edit

Places a variable selected in the window into edit mode. While in this state, you can change the value of that variable.

Format

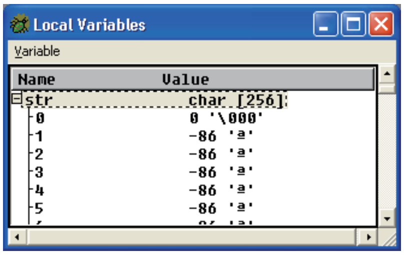
Choose the format in which you wish to display the variable value selected in the window.



Hex: Hexadecimal
Decimal: Signed decimal
Binary: Binary
Octal: Octal
Natural: Signed decimal (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.



```
Name                 Value
⊟str                 char [256]
 ├0                  0 '\000'
 ├1                  -86 'ª'
 ├2                  -86 'ª'
 ├3                  -86 'ª'
 ├4                  -86 'ª'
 ├5                  -86 'ª'
```

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.

When binary format is selected, the window displays the symbol values for number of digits equivalent to the symbol size.
Example: `int iSymbol = 0x1a55;`

The size of int type variable is 16 bits, so this symbol value is displayed as "`iSymbol  0001101001010101`".

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

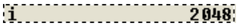
Click the [Close] button to close the window. The window can also be minimized.
Only one instance of the [Local Variables] window can be open at a time. Any attempt to open the [Local Variables] window while already open reactivates the [Local Variables] window and moves it to the front of the screen.

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:	Signed decimal (default)

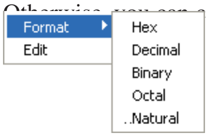
Select a variable whose display format you wish to change by clicking on its line in the window.



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.

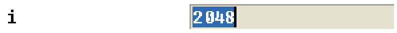


Rewriting variable values

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.



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.

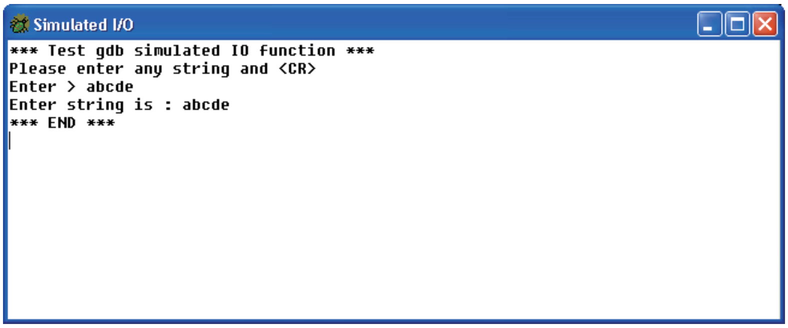
Entering a value exceeding the data size of the variable only applies the low-order effective bits and ignores the bits outside the valid range.

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.

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 `c33 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 `c33 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.

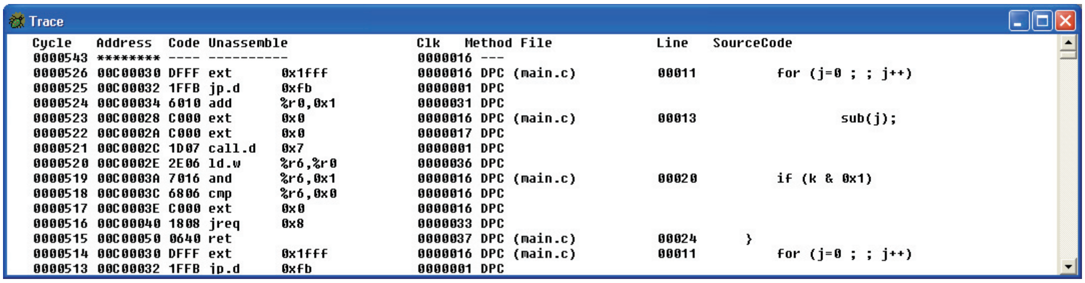
10.4.10 [Trace] Window

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 not available in debug monitor mode.

ICD2, ICD3 and ICD6 modes (PC trace)

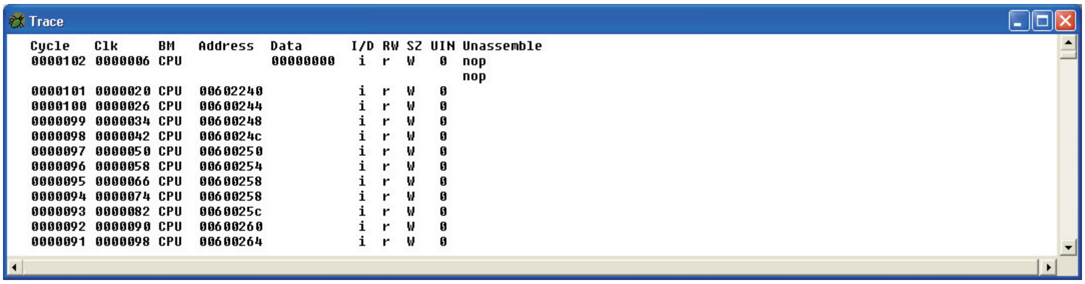


```
Cycle   Address  Code Unassemble              Clk     Method File            Line    SourceCode
0000543 ******** ---- ----------              0000016 ---
0000526 00C00030 DFFF ext        0x1fff       0000016 DPC (main.c)           00011          for (j=0 ; ; j++)
0000525 00C00032 1FFB jp.d       0xfb         0000001 DPC
0000524 00C00034 6010 add        %r0,0x1      0000031 DPC
0000523 00C00028 C000 ext        0x0          0000016 DPC (main.c)           00013                   sub(j);
0000522 00C0002A C000 ext        0x0          0000017 DPC
0000521 00C0002C 1D07 call.d     0x7          0000001 DPC
0000520 00C0002E 2E06 ld.w       %r6,%r0      0000036 DPC
0000519 00C0003A 7016 and        %r6,0x1      0000016 DPC (main.c)           00020          if (k & 0x1)
0000518 00C0003C 6806 cmp        %r6,0x0      0000016 DPC
0000517 00C0003E C000 ext        0x0          0000016 DPC
0000516 00C00040 1808 jreq       0x8          0000033 DPC
0000515 00C00050 0640 ret                     0000037 DPC (main.c)           00024      }
0000514 00C00030 DFFF ext        0x1fff       0000016 DPC (main.c)           00011          for (j=0 ; ; j++)
0000513 00C00032 1FFB jp.d       0xfb         0000001 DPC
```

In ICD2, ICD3 and ICD6 modes, the PC trace results can be read out from trace memory of the S5U1C33000H or S5U1C33001H for display in this window. Three types of PC trace method are supported: whole trace with or without overwrite and range trace.
The following contents are displayed:

- Number of run cycles
- Executed address, code, and disassembled content
- Number of clock cycles
- Method of determining PC
- Number of source lines and source code

ICD3 and ICD6 modes (On chip bus trace)



```
Cycle   Clk     BM  Address  Data     I/D RW SZ UIN Unassemble
0000102 0000006 CPU          00000000  i  r  W   0  nop
                                                    nop
0000101 0000020 CPU 00602240           i  r  W   0
0000100 0000026 CPU 00600244           i  r  W   0
0000099 0000034 CPU 00600248           i  r  W   0
0000098 0000042 CPU 0060024c           i  r  W   0
0000097 0000050 CPU 00600250           i  r  W   0
0000096 0000058 CPU 00600254           i  r  W   0
0000095 0000066 CPU 00600258           i  r  W   0
0000094 0000074 CPU 00600258           i  r  W   0
0000093 0000082 CPU 0060025c           i  r  W   0
0000092 0000090 CPU 00600260           i  r  W   0
0000091 0000098 CPU 00600264           i  r  W   0
```

In ICD3 and ICD6 modes, an on-chip bus trace supported by the C33 ADV Core is also possible, and the trace results can be displayed.
The contents displayed in normal mode (above display example) are listed below.

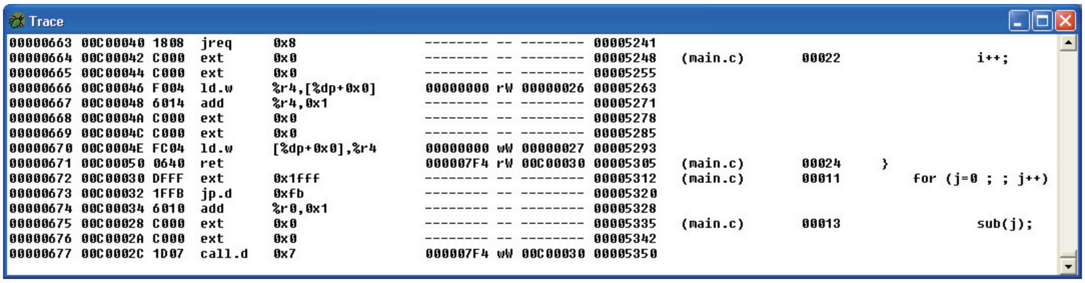
- Number of run and clock cycles
- Bus type
- Accessed address and data
- Content of the memory accessed (instruction/data, R/W, access size)
- USER_IN signal state
- Disassembled content (for instructions)

Listed below are the contents displayed in user mode (see Section 10.6.7).

- Number of run and clock cycles
- DTD and DTS signal states
- USER_IN signal state

PC trace information can also be displayed along with the above by setting a command for that purpose.

Simulator mode (PC trace)



```
00000663 00C00040 1808  jreq     0x8                 -------- -- -------- 00005241
00000664 00C00042 C000  ext      0x0                 -------- -- -------- 00005248   (main.c)      00022               i++;
00000665 00C00044 C000  ext      0x0                 -------- -- -------- 00005255
00000666 00C00046 F004  ld.w     %r4,[%dp+0x0]       00000000 rW 00000026 00005263
00000667 00C00048 6014  add      %r4,0x1             -------- -- -------- 00005271
00000668 00C0004A C000  ext      0x0                 -------- -- -------- 00005278
00000669 00C0004C C000  ext      0x0                 -------- -- -------- 00005285
00000670 00C0004E FC04  ld.w     [%dp+0x0],%r4       00000000 wW 00000027 00005293
00000671 00C00050 0640  ret                          000007F4 rW 00C00030 00005305   (main.c)      00024     }
00000672 00C00030 DFFF  ext      0x1fff              -------- -- -------- 00005312   (main.c)      00011          for (j=0 ; ; j++)
00000673 00C00032 1FFB  jp.d     0xfb                -------- -- -------- 00005320
00000674 00C00034 6010  add      %r0,0x1             -------- -- -------- 00005328
00000675 00C00028 C000  ext      0x0                 -------- -- -------- 00005335   (main.c)      00013               sub(j);
00000676 00C0002A C000  ext      0x0                 -------- -- -------- 00005342
00000677 00C0002C 1D07  call.d   0x7                 000007F4 wW 00C00030 00005350
```

In simulator mode, once the trace function is turned on (by the `c33 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 (`c33 tm` command option)
- Register content (`c33 tm` command option)

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.

Otherwise, the [Trace] window opens automatically when the `c33 td` command is executed in ICD2, ICD3, or ICD6 mode.

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

ICD2, ICD3 and ICD6 modes

Executing the target program clears the contents of the [Trace] window.

To display the latest contents after stopping the program midway, execute the `c33 td` command or temporarily close the [Trace] window, then open it again.

Simulator mode

When the trace function is turned on (by the `c33 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

In simulator mode, up to 255 lines can be scrolled in the window.

In ICD2/ICD3/ICD6 modes, all contents of S5U1C33000H or S5U1C33001H trace memory can be displayed.

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/gnu33/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]	U
`c33 rstc` *	[Source]	[Reset]-[Reset cold]	⟲
`c33 rsth` *	[Source]	[Reset]-[Reset hot]	⟲
`C33 savebreak` `C33 savewatch`	[Source]	[Control]-[Save breakpoints/Watch Symbols]	🗋

`Delete`
`C33 loadbreak`
`C33 loadwatch`
	[Source]	[Control]-[Load breakpoints/Watch Symbols]	📂

`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},`	[Memory]	Data entered from keyboard	–
`set {int}`			
`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 `\gnu33\userdefine.gdb`.

Contents of `userdefine.gdb` at shipment

`#Edit user command`
`#c33 rsth`   (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.
`\gnu33\utility\share\gdbtcl\images2\userdefine.gif`

[Reset]-[Reset cold]

Executes the command file `\gnu33\resetcold.gdb` that can be edited by the user.

Contents of `resetcold.gdb` at shipment

`c33 rstc`   (Cold-resets the CPU)

[Reset]-[Reset hot]

Executes the command file `\gnu33\resethot.gdb` that can be edited by the user.

Contents of `resethot.gdb` at shipment

`c33 rsth`   (Hot-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

The following shows a command file included in the simulated I/O sample. Only one command can be written per line.

Example: Filename = `simIO.cmd`

`c33 rpf 33104_1.par`	Sets memory map information.
`file simIO.elf`	Loads debug information.
`target sim`	Connects the target.
`set {char}0x4812d=0x59`	Removes boot vector address write protection.
`set {int}0x48134=0xc00000`	Sets the boot vector address.
`load`	Loads a program.
`c33 rsth`	Executes hot reset.
`c33 stdout 1 WRITE_FLASH WRITE_BUF stdout.txt`	Sets stdout.
`c33 stdin 1 READ_FLASH READ_BUF stdin.txt`	Sets stdin.
`break _exit`	Sets a software PC breakpoint.
`cont`	Executes the program.
`c33 stdout 2`	Clears stdout.
`c33 stdin 2`	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\gnu33\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 `--c33_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 `--c33_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) c33 log test.log`	Starts logging.
`:`	Log mode
`(gdb) c33 log`	Finishes logging.

After logging is started by the `c33 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 four connect modes, of which the mode used is set by the `target` command.

ICD2 mode

In this mode, the in-circuit debugger S5U1C33000H (ICD ver. 2) is used to perform debugging. The program is executed on the target board, with trace data sent to and saved in internal memory of the S5U1C33000H.

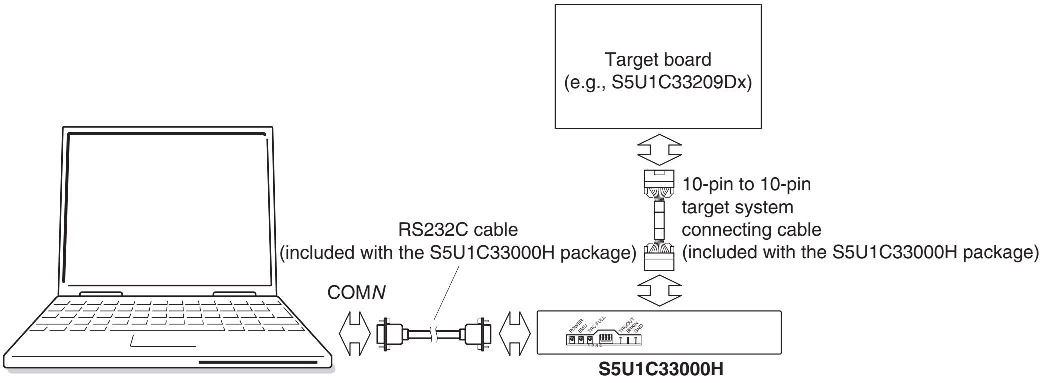

Target board
(e.g., S5U1C33209Dx)
10-pin to 10-pin
target system
connecting cable
(included with the S5U1C33000H package)
RS232C cable
(included with the S5U1C33000H package)
COM*N*
POWER
EMU
TRC FULL
1 2 3 4
TRGOUT
BRKIN
GND
S5U1C33000H

Figure 10.6.1.1 Example of debugging system using the S5U1C33000H

Specification method

Command: `(gdb)` **`target icd comN`**

`comN`: Specify the serial port used (com1 to com9).

Specification in **IDE**:

Select "ICD Ver2" from the [Debugger:] combo box in the [Create a simple startup command] dialog box to generate a startup command file. Additionally, use this dialog box to select a COM port and baud rate.

To start in ICD2 mode, make sure the S5U1C33000H and target board are connected correctly, and that the power for these units is turned on. To use the trace function, also make sure DIP switch SW4 of the S5U1C33000H is left open.
In ICD2 mode, you cannot use the flash writer function.
For details on how to use the S5U1C33000H, refer to the "S5U1C33000H Manual (S1C33 Family In-Circuit Debugger)".

ICD3 mode

In this mode, the in-circuit debugger S5U1C33001H1100 (ICD ver. 3) or S5U1C33001H1200 (ICD ver. 4) is used to perform debugging. The program is executed on the target board, with trace data sent to and saved in internal memory of the S5U1C33001H1100/1200.

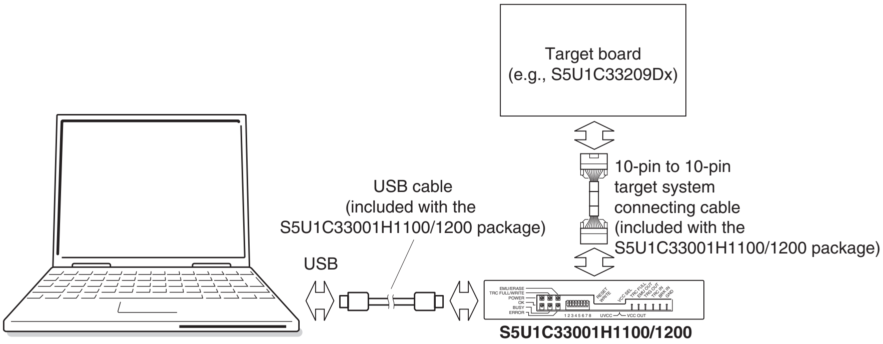

Target board
(e.g., S5U1C33209Dx)
USB cable
(included with the
S5U1C33001H1100/1200 package)
10-pin to 10-pin
target system
connecting cable
(included with the
S5U1C33001H1100/1200 package)
USB
EMU/ERASE
TRC FULL/WRITE
POWER
OK
BUSY
ERROR
1 2 3 4 5 6 7 8
RESET
WRITE
VCC SEL
TRC FULL
EMU OUT
TRG OUT
TRC IN
BRK IN
GND
UVCC
VCC OUT
S5U1C33001H1100/1200

Figure 10.6.1.2 Example of debugging system using the S5U1C33001H1100/1200 (ver. 3/ver. 4)

Specification method

Command: `(gdb) target icd usb`

Specification in **IDE**:

Select "ICD Ver3/Ver4" from the [Debugger:] combo box in the [Create a simple startup command] dialog box to generate a startup command file.

To start in ICD3 mode, make sure the S5U1C33001H1100/1200 and target board are connected correctly, and that the power for these units is turned on. To use the trace function, also make sure DIP switch SW4 of the S5U1C33001H1100/1200 is left open. For details on how to use the S5U1C33001H1100/1200, refer to the "S5U1C33001H Manual (S1C33 Family In-Circuit Debugger)".

ICD6 mode

In this mode, the in-circuit debugger S5U1C33001H1400 (ICD ver. 6) is used to perform debugging. The program is executed on the target board, with trace data sent to and saved in internal memory of the S5U1C33001H1400.

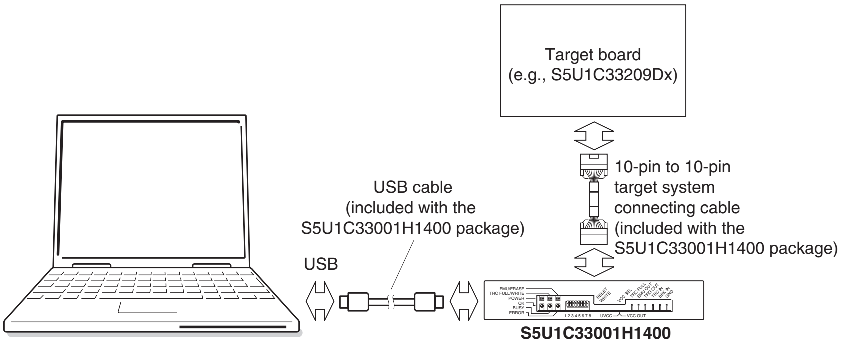

Target board
(e.g., S5U1C33209Dx)
USB cable
(included with the
S5U1C33001H1400 package)
10-pin to 10-pin
target system
connecting cable
(included with the
S5U1C33001H1400 package)
USB
EMU/ERASE
TRC FULL/WRITE
POWER
OK
BUSY
ERROR
1 2 3 4 5 6 7 8
RESET
WRITE
VCC SEL
TRC FULL
EMU OUT
TRG OUT
TRC IN
BRK IN
GND
UVCC
VCC OUT
S5U1C33001H1400

Figure 10.6.1.3 Example of debugging system using the S5U1C33001H1400 (ver. 6)

Specification method

Command: `(gdb) target icd6 usb`

Specification in **IDE**:

Select "ICD Ver6" from the [Debugger:] combo box in the [Create a simple startup command] dialog box to generate a startup command file.

To start in ICD6 mode, make sure the S5U1C33001H1400 and target board are connected correctly, and that the power for these units is turned on. For details on how to use the S5U1C33001H1400, refer to the "S5U1C33001H1400 Manual (S1C33 Family In-Circuit Debugger)".

Debug monitor (MON) mode

In debug monitor mode, the program is debugged on the target board connected to the PC via the S5U1C330M1D1. The target board must have the debug monitor S5U1C330M2S installed on it.

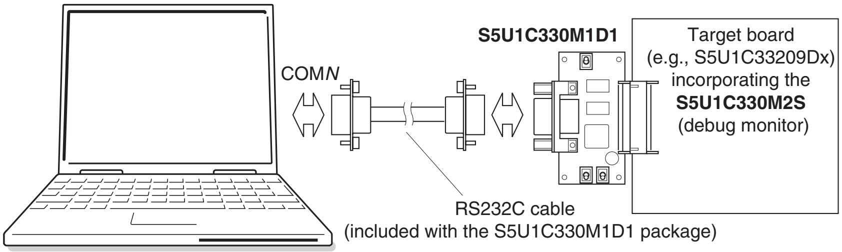


Figure 10.6.1.4 Example of debugging system using the S5U1C330M1D1 and S5U1C330M2S

Specification method

Command: `(gdb)` **`target icd comN`**

`comN`: Specify the serial port used (com1 to com9).

Specification in **IDE**:

Select "MON" from the [Debugger:] combo box in the [Create a simple startup command] dialog box to generate a startup command file. Additionally, use this dialog box to select a COM port and baud rate.

To start in debug monitor mode, make sure the S5U1C330M1D1 and target board are connected correctly, and that the power for these units is turned on. The debug monitor on the target board must also be up and running.

In debug monitor mode, you cannot use the trace function, flash writer function, and [Stop] button-actuated forcible breaks. For details about the debug monitor, refer to the "S1C33 Family Debug Monitor Operation Manual".

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 S5U1C33000H or S5U1C33001H-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 method in simulator mode is different from those in ICD2, ICD3 and ICD6 modes. The flash writer function also cannot be used.

In simulator mode, each area is initialized as shown below.

- RAM: 0xaa
- ROM: 0xff
- I/O: 0x00

Precautions on debugging in ICD and debug monitor modes

When the program being executed is made to break in the S5U1C33000H or S5U1C33001H, the CPU operating clock switches to the high-speed clock (OSC3), and clocks supplied to all peripheral functions on the S1C33 chip (except DRAM refresh) are turned off during a break.

When the program is executed and made to break in the S5U1C330M2S, the same state as described above is momentarily entered, with the previous state restored immediately after.

Therefore, systems not using the OSC3 clock cannot be debugged.

Moreover, when operating with the low-speed clock (OSC1, 32 kHz), with the OSC3 oscillator circuit turned off, the OSC3 clock starts oscillating the moment the program being executed breaks. The CPU may operate erratically due to unstable clock oscillation. Avoid breaking program execution while the OSC3 oscillator remains idle, as well as during SLEEP mode.

When using the S5U1C33001H to perform debugging (ICD3 or ICD6 mode), always be sure to quit the debugger before powering off the S5U1C33001H. If the S5U1C33001H is powered off while running the debugger, you will not be able to reconnect it. In this case, you need to restart your computer.

Connect modes and the supported target CPU

Table 10.6.1.1 lists the target CPUs supported in each connect mode.

Table 10.6.1.1 Connect mode and support target CPU

Connect mode	C33 STD	C33 PE	S1C33401
ICD2	○	×	×
ICD3	○	○	○
ICD6	○	○	○
Debug monitor	○	×	×
Simulator	○	○	○

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 **`c33 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 `c33 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) c33 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) c33 rsth            (Executes hot reset.)
```

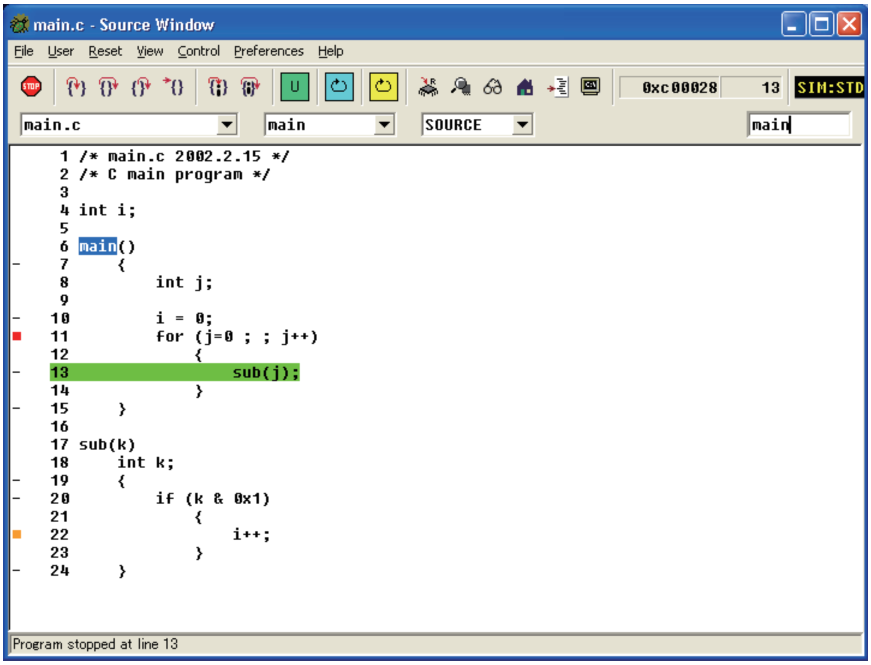
To debug a program written in target ROM, there is no need to execute the `load` command. In this case, the `file` command can also be used to load debugging information for source-level debugging.

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/C++ source and assembly source in the [Source] window.


main.c - Source Window
File
User
Reset
View
Control
Preferences
Help
STOP
U
0xc00028
13
SIM:STD
main.c
main
SOURCE
main

```
 1 /* main.c 2002.2.15 */
 2 /* C main program */
 3
 4 int i;
 5
 6 main()
 7     {
 8         int j;
 9
10         i = 0;
11         for (j=0 ; ; j++)
12             {
13                 sub(j);
14             }
15     }
16
17 sub(k)
18     int k;
19     {
20         if (k & 0x1)
21             {
22                 i++;
23             }
24     }
```

Program stopped at line 13

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. Half word and word 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, half words, or words.

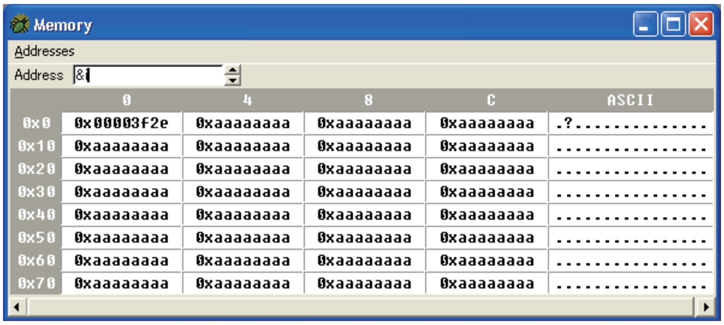
Memory dump (`x /b`, `x /h`, `x /w` commands)

Dumps memory contents for a specified amount of data from a specified address for display on the screen.

Example: Memory dump for 16 half words 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 0xdff8 0xdfff 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".



Address &i

	0	4	8	C	ASCII
0x0	0x00003f2e	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	.?..............
0x10	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	....
0x20	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	....
0x30	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	....
0x40	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	....
0x50	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	....
0x60	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	....
0x70	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	0xaaaaaaaa	....

Entering data (`set {char}`, `set {short}`, `set {int}` commands)

Writes specified data to a specified address.

Data can also be entered or changed in the [Memory] window.

Example: Setting `int i` to 0x55555555

```
(gdb) set {int}&i=0x55555555
```

Rewriting a specified area (`c33 fb`, `c33 fh`, `c33 fw` commands)

Rewrites all of a specified area with specified data.

Example: Writing 4 words of 0x1 to addresses 0x0 through 0xf

```
(gdb) c33 fw 0x0 0xf 0x1
Start address = 0x0, End address = 0xc, Fill data = 0x1 .....done
```

Copying a specified area (`c33 mvb`, `c33 mvh`, `c33 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) c33 mvb 0x0 0x7 0x8
Start address = 0x0, End address = 0x7, Destination address = 0x8 .....done
```

Saving memory contents (`c33 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) c33 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 (`c33 readmd` command)

ICD3/ICD6 mode only

In an ordinary memory read by the `x` command or `c33 df` command, data is always read out in units of bytes, regardless of accessed data size. If this read method becomes inconvenient, use the **`c33 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) c33 readmd 1
```

Notes:
- When memory contents are read from addresses in the internal I/O area containing no registers, etc., the last data read out by the CPU is displayed (due to chip specifications).
- The S5U1C33000H and S5U1C33001H are designed to read data from memory 8 bytes at a time. For this reason, up to 7 bytes of data may be read ahead from the address specified by a command. When accessing I/O memory, pay attention to registers for which status is changed by read operation.
- When writing data to internal ROM emulation memory of the S5U1C33*XXX*E by using the S5U1C33000H/S5U1C33001H or S5U1C330M2S, do not use commands that write data in bytes (e.g., `set {char}`, `c33 fb`, `c33 mvb`). Always be sure to use commands that write data in half word or larger units (e.g., `set {short}`, `set {int}`, `c33 fh`, `c33 fw`, `c33 mvh`, `c33 mvw`). The file load command (`load`) writes data in units of half words.

Variable list

A list of variables can be displayed.

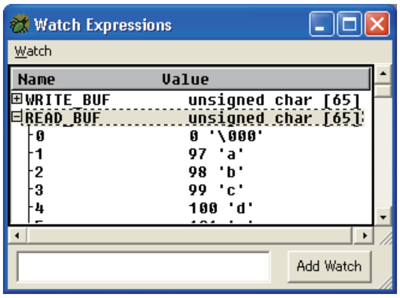
Displaying global variables (`info var`, `print` commands)

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".



Name	Value
⊞WRITE_BUF	unsigned char [65]
⊟READ_BUF	unsigned char [65]
0	0 '\000'
1	97 'a'
2	98 'b'
3	99 'c'
4	100 'd'

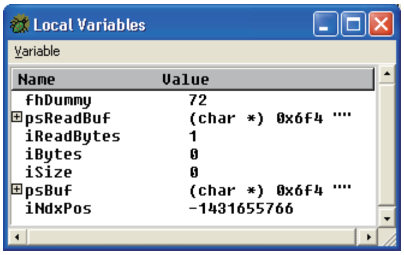
Displaying local variables (`info locals` command)

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".



Name	Value
fhDummy	72
⊞psReadBuf	(char *) 0x6f4 ""
iReadBytes	1
iBytes	0
iSize	0
⊞psBuf	(char *) 0x6f4 ""
iNdxPos	-1431655766

Register operation

The following operations can be performed in registers.

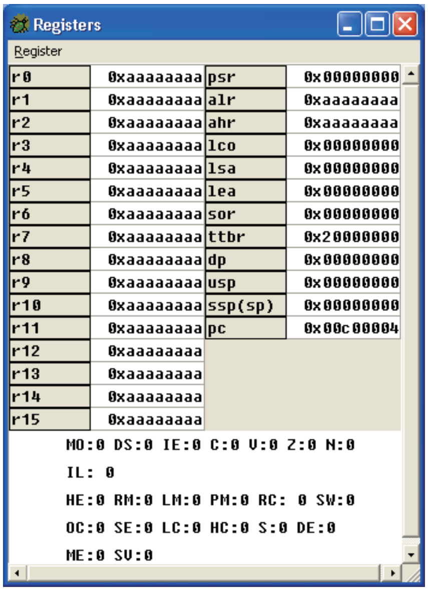
Displaying registers (`info reg` command)

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 (for the C33 ADV Core)

```
(gdb) info reg
r0             0xd20          3360
r1             0xaaaaaaaa     -1431655766
r2             0xaaaaaaaa     -1431655766
r3             0xaaaaaaaa     -1431655766
r4             0x690          1680
r5             0xaaaaaaaa     -1431655766
r6             0x0            0
r7             0xaaaaaaaa     -1431655766
r8             0xaaaaaaaa     -1431655766
r9             0xaaaaaaaa     -1431655766
r10            0xaaaaaaaa     -1431655766
r11            0xaaaaaaaa     -1431655766
r12            0xaaaaaaaa     -1431655766
r13            0xaaaaaaaa     -1431655766
r14            0xaaaaaaaa     -1431655766
r15            0x0            0
psr            0x2            2
sp             0x7f8          2040
alr            0x0            0
ahr            0x0            0
lco            0x0            0
lsa            0x0            0
lea            0x0            0
sor            0x0            0
ttbr           0x20000000     536870912
dp             0x0            0
idir           0x4000000      67108864
dbbr           0x60000        393216
usp            0x0            0
ssp            0x0            0
pc             0xc00030       12582960
```

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".



r0	0xaaaaaaaa	psr	0x00000000
r1	0xaaaaaaaa	alr	0xaaaaaaaa
r2	0xaaaaaaaa	ahr	0xaaaaaaaa
r3	0xaaaaaaaa	lco	0x00000000
r4	0xaaaaaaaa	lsa	0x00000000
r5	0xaaaaaaaa	lea	0x00000000
r6	0xaaaaaaaa	sor	0x00000000
r7	0xaaaaaaaa	ttbr	0x20000000
r8	0xaaaaaaaa	dp	0x00000000
r9	0xaaaaaaaa	usp	0x00000000
r10	0xaaaaaaaa	ssp(sp)	0x00000000
r11	0xaaaaaaaa	pc	0x00c00004
r12	0xaaaaaaaa		
r13	0xaaaaaaaa		
r14	0xaaaaaaaa		
r15	0xaaaaaaaa		

```
MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
IL: 0
HE:0 RM:0 LM:0 PM:0 RC: 0 SW:0
OC:0 SE:0 LC:0 HC:0 S:0 DE:0
ME:0 SV:0
```

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.

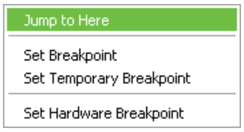
[Continue] button

∗ 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, except in debug monitor mode)
- Other causes of break generated

[Stop] button

∗ 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.

[Next] button

To execute the `nexti` command:
- Choose [Next Asm Inst] from the [Control] menu.
- Click the [Next Asm Inst] button.

[Next Asm Inst] button

To execute the `finish` command:
- Choose [Finish] from the [Control] menu.
- Click the [Finish] button.

[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, except in debug monitor mode)
- Other causes of break than those set by the user

During single-stepping, the program does not stop at PC or data breakpoints.

[Stop] button

* 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. For example, by calling the internal functions of the S1C33 Family real-time OS, you can get various debugging information. For details, refer to the "S1C33 Family Real-Time OS Operation Manual".

Example: Outputting a list of real-time OS management data

```
(gdb) call mng
```

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

In ICD2, ICD3, ICD6, or debug monitor mode, 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.

Standby mode and interrupts are not supported in simulator mode. The program is made to break when the `halt` or `slp` instruction is executed.

Measuring the execution cycles/execution time

In ICD2, ICD3, ICD6, or simulator mode, you can measure the program execution cycles or time. This measurement facility is not supported in debug monitor mode.

Execution counter and measurement mode

The S5U1C33000H contains one 29-bit execution counter; the S5U1C33001H contains one 31-bit execution counter. By using the **`c33 clockmd`** command, this counter can be set for measurement in units of execution time (in units of µs or seconds) or the number of cycles executed. In simulator mode, the counter only counts the number of cycles executed.

Table 10.6.5.1 Units of measurement and accuracy of execution counters

Measurement unit	Accuracy	
	ICD2/ICD3/ICD6 mode	Simulator mode
Measurement of execution time (in units of seconds)	±2 µs *1	–
Measurement of execution time (in units of µs)	±100 ns *1	–
Measurement of number of cycles (in units of cycles)	±16 cycles *1	±0 cycles

*1 Due to `continue` and break overheads, prefetch cycles are added for two more instructions.

The following shows the maximum values that can be measured by the execution counters of the S5U1C33000H and S5U1C33001H.

When measured in units of seconds:	About 9 minutes for the S5U1C33000H About 36 minutes for the S5U1C33001H
When measured in units of µs:	About 27 seconds for the S5U1C33000H About 1.8 minutes for the S5U1C33001H
When measured in units of cycles:	2,147,483,644 cycles for the S5U1C33000H 8,589,934,588 cycles for the S5U1C33001H

Displaying measurement results

The measurement results can be displayed in the [Console] window by using the **`c33 clock`** command.
Example:

```
(gdb) c33 clock
          330 cycle
```

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.

In reset mode (set by the `c33 clockmd` command), the counter is reset each time the program is executed.
In cases where the counter is set to reset mode and the program executed continuously, the counter is reset when the program is started by entering an execution command (`continue` or `until`) and continues counting until the program terminates (or made to break). When the program is single-stepped, the counter is reset when the program is started by entering a step command (e.g., `step` or `next`) and continues counting until the program finishes executing the specified number of steps. When only one step of the program is executed or executed by using a toolbar button or menu command, the counter is reset each time the program is executed one step.

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 `c33 clockmd` command
- When the program is started while the counter is set to reset mode
- When the CPU is cold-reset or hot-reset

Even in integrating mode, executing the program by a command other than `continue` or `until` (e.g., `step` or `next`) will reset the execution counter. If the execution time must be measured, use the `continue` or `until` command.

Resetting the CPU

The CPU is cold-reset by the **`c33 rstc`** command or hot-reset by the **`c33 rsth`** command.
These commands can also be executed using the menu or toolbar buttons in the [Source] window.

To execute the `c33 rstc` command:
(`\gnu33\resetcold.gdb`)

- Choose [Reset cold] from the [Reset] menu.
- Click the [Reset cold] button.

[Reset cold] button

To execute the `c33 rsth` command:
(`\gnu33\resethot.gdb`)

- Choose [Reset hot] from the [Reset] menu.
- Click the [Reset hot] button.

[Reset hot] button

Note: These menu commands/buttons execute the respectively 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-R15:` `0xaaaaaaaa`
`PC:` Boot address (indicated by content of 0xc00000)
`SP:` `0x0aaaaaa8` (C33 STD, C33 PE)
`0x00000000` (C33 ADV)
`PSR:` `0x00000000`
`AHR, ALR:` `0xaaaaaaaa`
`LCO:` `0x00000000` (C33 ADV)
`LSA:` `0x00000000` (C33 ADV)
`LEA:` `0x00000000` (C33 ADV)
`SOR:` `0x00000000` (C33 ADV)
`TTBR:` `0x20000000` (C33 ADV, C33 PE)
`DP:` `0x00000000` (C33 ADV)
`IDIR:` `0x40000000` (C33 ADV)
`0x60000000` (C33 PE)
`DBBR:` `0x00060000` (C33 ADV, C33 PE)
`USP:` `0x00000000` (C33 ADV)
`SSP:` `0x00000000` (C33 ADV)

The registers are shared by all cores unless otherwise noted. TTBR specifies the PC value (boot address) to be initialized at hot reset.

(2) The execution counter in the S5U1C33000H or S5U1C33001H is cleared to 0.

(3) The [Source] and [Registers] windows are updated.
Because the PC is set to the boot address, the [Source] window redisplays the program beginning with that address. The [Registers] window reappears with the initialized values.

Memory contents are not changed.

Note: The `c33 rstc` command functions differently in connect mode.

ICD2/ICD3/ICD6 modes:	The processing above is performed and the S1C33 chip is reset. The target board is not reset. If the target was operating in free-running mode when the `c33 rstc` command was executed, a forcible break is applied to the target before resetting it. When the target connected to the S5U1C33000H or S5U1C33001H is reset, the target system is placed in free-running mode, in which case the target can be made to stop running by the `c33 rstc` command.
Debug monitor mode:	The `c33 rstc` command functions the same way as the `c33 rsth` command. It neither resets the S1C33 chip nor initializes TTBR.
Simulator mode:	The boot address is determined by MCU/MPU specifications in a parameter file. In simulator mode when debugging for C33 ADV mode, if no map is set for one byte or more from address 0x20000000 in a parameter file, the PC value is initialized to the address indicated by the content of 0xc00000.

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, except in debug monitor mode).
- An illegal attempt is made to access memory, etc.

Command-actuated breaks

The debugger **gdb** supports the following six types of breaks for which break conditions can be set by a command:

1. Software PC break
2. Hardware PC break
3. Data break
4. On chip area (CE) break (C33 ADV)
5. On chip bus break (C33 ADV)
6. Lapse of time break (ICD6)

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. The total 200 breakpoints mentioned above include both types of breaks.

`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 `push` 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.

`ext` *xxxx*	... Can be set.	`jr*.d` *xxxx*	... Can be set.
`ext` *xxxx*	... 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 `c33 rpf` command as it checks each address to determine whether a software PC breakpoint can be set. Unless the `c33 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 debug mode of the C33 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 up to two address locations.
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".

Notes:
- Hardware PC breaks are disabled when an area trace in ICD2, ICD3, or ICD6 mode remains set.
- When you set a hardware 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.

`ext` *xxxx*	... Can be set.	`jr*.d` *xxxx*	... Can be set.
`ext` *xxxx*	... Cannot be set.	Delayed instruction	... Cannot be set.
Extended instruction	... Cannot be set.		

Data breaks (`watch`, `rwatch`, and `awatch` commands)

This type of break occurs when a specified memory address is accessed.

`watch` command: This command sets a data breakpoint that causes the program to break when it writes data to that location.
Example: To set a data write breakpoint at the address of variable `i`

```
(gdb) watch i
Hardware watchpoint 1: i
```

`rwatch` command: This command sets a data breakpoint that causes the program to break when it reads data from that location.
Example: To set a data read breakpoint at the address of variable `i`

```
(gdb) rwatch i
Hardware read watchpoint 2: i
```

`awatch` command: This command sets a data breakpoint that causes the program to break when it writes data to or reads data from that location.
Example: To set a data read/write breakpoint at address 0x0

```
(gdb) awatch *0x0
Hardware access (read/write) watchpoint 3: *0
```

A data break occurs after the program has finished accessing a specified memory address for read or write as specified. When this break occurs, the debugger waits for command input after displaying the message shown below.
Example 1: For a break at a data write breakpoint

```
(gdb) continue
Continuing.
Hardware watchpoint 1: i

Old value = 0
New value = 1
sub (k=1) at ./main.c:24
```

Example 2: For a break at a data read breakpoint

```
(gdb) continue
Continuing.
Hardware read watchpoint 2: i

Value = 2
0x00c00042 in sub (k=1) at ./main.c:22
```

Example 3: For a break at a data read/write breakpoint

```
(gdb) continue
Continuing.
Hardware access (read/write) watchpoint 3: *0

Value = 2
sub (k=1) at ./main.c:24
```

Data breakpoints can also be set from the [Source] window.
Place the cursor on a global variable name and right-click to display the popup menu. Select [Set Read/Write Data Breakpoint at <*symbol*>] from the popup menu to set a data breakpoint at which the program breaks upon access to the variable for read/write. If you want the program to break upon access for only read or write, select [Set Read Data Breakpoint at <*symbol*>] or [Set Write Data Breakpoint at <*symbol*>], respectively. Select [Delete Data Breakpoint at <*symbol*>] to clear the data breakpoint set.
For how to use the functions of the [Source] window, see Section 10.4.2, "[Source] Window".

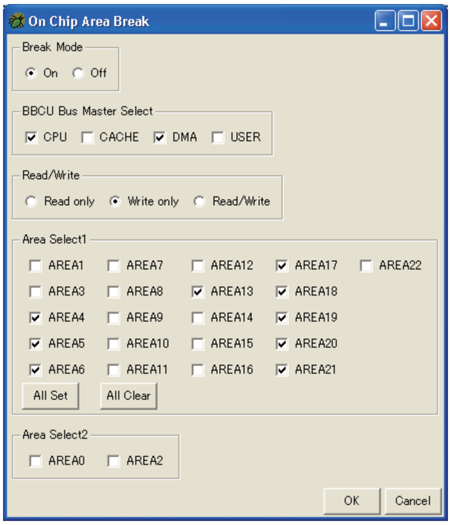
On chip area (CE) break (`c33 oab` command) ICD3/ICD6 mode, C33 ADV Core only

This break uses the features of the C33 ADV Core to cause the program to break when a specified CE area is accessed. The BBCU bus master, CE area, and read/write conditions may be specified.

Example 1: Set to break when the DMA or HBCU writes to the CE area 4–6, 13, or 17–21

```
(gdb) c33 oab on 0x05 w 0x3e2070
Area break on
BBCU bus master select : DMA,CPU
Read/Write             : Write
Area                   : 4,5,6,13,17,18,19,20,21
```

In addition to the above command input, it's possible to specify an on-chip area break from the [On Chip Area Break] dialog box shown below (select [Area Break] from the [View] menu to display the dialog box).



Break Mode
● On ○ Off

BBCU Bus Master Select
☑ CPU ☐ CACHE ☑ DMA ☐ USER

Read/Write
○ Read only ● Write only ○ Read/Write

Area Select1
☐ AREA1 ☐ AREA7 ☐ AREA12 ☑ AREA17 ☐ AREA22
☐ AREA3 ☐ AREA8 ☑ AREA13 ☑ AREA18
☑ AREA4 ☐ AREA9 ☐ AREA14 ☑ AREA19
☑ AREA5 ☐ AREA10 ☐ AREA15 ☑ AREA20
☑ AREA6 ☐ AREA11 ☐ AREA16 ☑ AREA21

Area Select2
☐ AREA0 ☐ AREA2

For more information on these settings, refer to the section on the `c33 oab` command.

Example 2: Clear the on-chip area (CE) break

```
(gdb) c33 oab off
```

An on-chip area (CE) break occurs after the completion of a specified memory access to a specified area. When this break occurs, the debugger stands by for command input after displaying a message similar to the one shown below.

Example:

```
(gdb) continue
Continuing.
Hardware on chip Area Break!
<<<Area Break Status>>>
Bus master : HBCU(CPU)
Access     : Write
Area       : 8
Data type  : Data
Data size  : Word
```

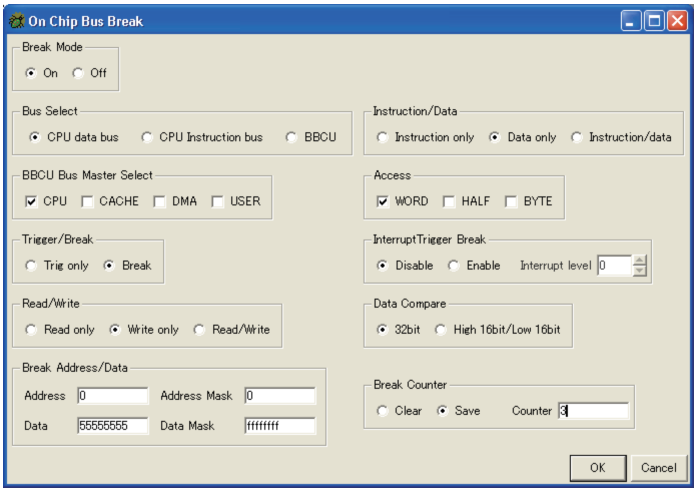
On chip bus break (`c33 obb` command) ICD3/ICD6 mode, C33 ADV Core only

This break uses the features of the C33 ADV Core to cause the program to break when a specified bus access occurs. You can specify the type of bus, BBCU bus master, address, data type and value, read/write conditions, and access counts. You can also break program execution by specifying a generated interrupt level.

Example 1: The program being executed is made to break when a word data 0x55555555 is written through the CPU system data bus three times. The bus data is compared in 32-bit size.

```
(gdb) c33 obb on 0 1 1 w 1 0x04 0 0 0 0 0 0x55555555 0xffffffff 3 0
Bus break on
Bus select                : CPU data bus
BBCU bus master select    : ---
Trigger/Break             : Break
Read/Write                : Write
Instruction/Data          : Data
Access size               : Word
Break by interrupt level  : Disable
Interrupt level           : 0
Data compare              : 32bit
Address                   : 0
Address mask              : 0
Data                      : 0x55555555
Data mask                 : 0xffffffff
Break counter             : 3
Break counter Clear/Save  : Save
```

In addition to the above command input, it's possible to specify an on-chip bus break from the [On Chip Bus Break] dialog box shown below (select [Bus Break] from the [View] menu to display the dialog box).


On Chip Bus Break

Break Mode: ● On ○ Off

Bus Select: ● CPU data bus ○ CPU Instruction bus ○ BBCU

Instruction/Data: ○ Instruction only ● Data only ○ Instruction/data

BBCU Bus Master Select: ☑ CPU ☐ CACHE ☐ DMA ☐ USER

Access: ☑ WORD ☐ HALF ☐ BYTE

Trigger/Break: ○ Trig only ● Break

InterruptTrigger Break: ● Disable ○ Enable Interrupt level 0

Read/Write: ○ Read only ● Write only ○ Read/Write

Data Compare: ● 32bit ○ High 16bit/Low 16bit

Break Address/Data: Address 0 Address Mask 0 Data 55555555 Data Mask ffffffff

Break Counter: ○ Clear ● Save Counter 3

For more information on these settings, refer to the section on the `c33 obb` command.

Example 2: Clear the on-chip bus break

```
(gdb) c33 obb off
```

An on-chip bus break occurs after a specified bus access is completed. When this break occurs, the debugger stands by for command input after displaying a message similar to the one shown below.

```
(gdb) continue
Continuing.
Hardware on chip Bus Break!
```

Lapse of time break (`c33 timebrk` command) ICD6 mode

This function forcibly breaks program execution when the specified time (1 to 300000 milliseconds) has elapsed after the program is started. This break can be used only in ICD6 mode.

Example 1: To break program execution after 1 second from starting

```
(gdb) c33 timebrk 1000
timer break on. [1000 ms]
```

Example 2: To disable lapse of time break

```
(gdb) c33 timebrk 0
timer break off.
```

When a break time has been set once, the lapse of time break is effective until it is disabled with "`c33 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.

Breakpoint control (software PC break, hardware PC break, data break)

When software PC breakpoints, hardware PC breakpoints, or data 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
3   hw watchpoint  keep y              i
  breakpoint already hit 6 times
```

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".

Note: If the CPU is cold-reset while running the target program in ICD2, ICD3, ICD6, or debug monitor mode, the hardware PC breaks supported by the chip (including internally used temporary breaks) and data breaks are cleared, and breaks will no longer occur. If the program is made to break once by another cause, the breakpoints are set back again, but otherwise remain cleared.

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.

Note: This forcible break cannot be used in debug monitor mode.

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 `c33 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.
```

Execution of `halt` or `slp` instruction

The program is made to break when executing the `halt` or `slp` instruction.

```
Break by sleep or halt.
```

10.6.7 Trace Functions

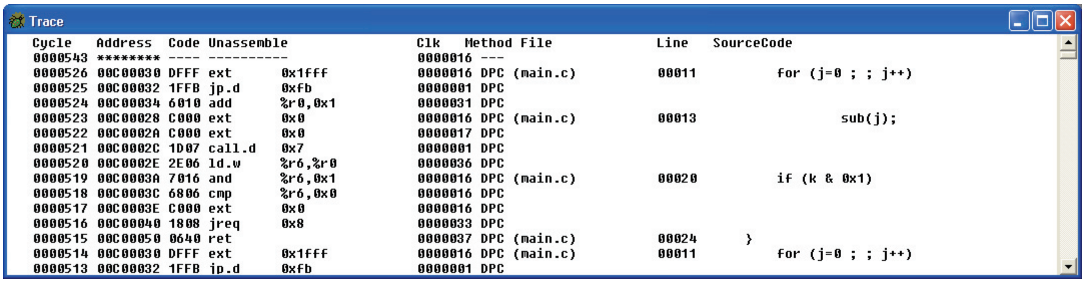
The debugger has a function to trace program execution.
Note that this function varies with each connect mode, so does the method of operation.

Note: The trace function cannot be used in debug monitor mode.

PC trace function in ICD2, ICD3 and ICD6 modes

Trace memory and PC trace information

The S5U1C33000H and S5U1C33001H both contain trace memory capable of storing 131,072 and 1,048,575 cycles of trace information, respectively. Information about instruction execution cycles are acquired in this memory by using, for example, the debugging signals output by the S1C33 chip. The acquired information is displayed in the [Trace] window by a command, and can also be written to a file.



```
Cycle   Address  Code Unassemble                Clk   Method File              Line   SourceCode
0000543 ******** ---- ----------                0000016 ---
0000526 00C00030 DFFF ext      0x1fff           0000016 DPC (main.c)           00011          for (j=0 ; ; j++)
0000525 00C00032 1FFB jp.d     0xfb             0000001 DPC
0000524 00C00034 6010 add      %r0,0x1          0000031 DPC
0000523 00C00028 C000 ext      0x0              0000016 DPC (main.c)           00013                  sub(j);
0000522 00C0002A C000 ext      0x0              0000017 DPC
0000521 00C0002C 1D07 call.d   0x7              0000001 DPC
0000520 00C0002E 2E06 ld.w     %r6,%r0          0000036 DPC
0000519 00C0003A 7016 and      %r6,0x1          0000016 DPC (main.c)           00020          if (k & 0x1)
0000518 00C0003C 6806 cmp      %r6,0x0          0000016 DPC
0000517 00C0003E C000 ext      0x0              0000016 DPC
0000516 00C00040 1808 jreq     0x8              0000033 DPC
0000515 00C00050 0640 ret                       0000037 DPC (main.c)           00024      }
0000514 00C00030 DFFF ext      0x1fff           0000016 DPC (main.c)           00011          for (j=0 ; ; j++)
0000513 00C00032 1FFB jp.d     0xfb             0000001 DPC
```

The contents of trace information are listed below.

Cycle:	Trace cycles (in decimal) The last information saved in trace memory is 000000.
Address:	Instruction addresses executed by the CPU (in hexadecimal)
Code:	Instruction codes executed by the CPU (in hexadecimal)
Unassemble:	Disassembled contents of instruction code
Clk:	Number of clock cycles executed This information can be displayed as cumulative clock cycles since a trace began or as clock cycles for each instruction executed by using a parameter in the `c33 tm` command. (Shown above are cumulative clock cycles.)
Method:	Method of trace analysis (or how to get instruction address) `SPC`: Analysis by initial PC `TRG`: Analysis by trigger address `DPC`: Analysis by DPCO signal `RET`: Analysis by `call` and `ret` statements `MAP`: Analysis by map information `RTI`: Analysis by `reti` statement `---`: Analysis impossible
File:	Source file name that includes executed instructions
Line:	Source line numbers
SourceCode:	Source codes

PC trace modes and trace conditions (`c33 tm` command)

Three trace modes are available to choose from depending on the trace area. The **`c33 tm`** command is used to select trace modes.

All trace mode (with overwrite, without overwrite)

Trace begins when the program starts running. You can choose "with overwrite" or "without overwrite" as a trace condition. Choosing "with overwrite" saves trace information in trace memory until the program breaks. When trace memory is full in the middle, the old data is overwritten beginning with the oldest. Choosing "without overwrite" terminates trace when trace memory is full.

You also can choose whether the number of clock cycles (Clk) in trace information is counted individually for each instruction or counted cumulatively for all instructions executed.

Example 1: To set all trace mode without overwrite, with cumulative execution cycles displayed

```
(gdb) c33 tm 1 1
```

Example 2: To set all trace mode with overwrite, with execution cycles for each instruction displayed

```
(gdb) c33 tm 2 2
```

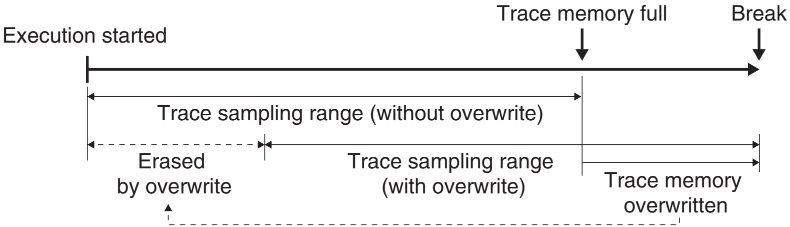


Figure 10.6.7.1 All trace mode

Area trace mode

A trace area is specified so that trace information is saved only when the program executes the specified trace area. The program can be made to break at the end-of-trace address. Moreover, a time measurement condition (whether to measure the execution start-to-break period or area trace time) can be specified. You also can choose whether the number of clock cycles (Clk) in trace information is counted individually for each instruction or counted cumulatively for all instructions executed.

Example: To set area trace mode from address 0x600000 to 0x600100, with cumulative execution cycles displayed, break after trace selected, and trace area execution time measured

```
(gdb) c33 tm 3 1 0x600000 0x600100 2 2
```

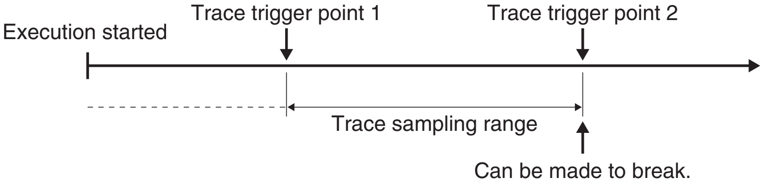


Figure 10.6.7.2 Area trace mode

Displaying PC trace information (`c33 td` command)

The **`c33 td`** command allows you to display the trace information currently saved in trace memory in the [Trace] window. You also can specify the range of trace information to display by a cycle number. Otherwise, you can execute the `c33 ts` command to display only the searched information.

Example: Displays a range of trace information currently in trace memory that is equivalent to 10K clock cycles

```
(gdb) c33 td
```

PC trace information search mode (`c33 ts` command)

The **`c33 ts`** command sets a given trace to search mode where you can display or save trace information for only several cycles, centered on the one in which the instruction at a specified address was executed.

Example: Set a trace to search mode where trace information for only the cycle in which the instruction at address 0x20100 was executed, with the five cycles before and after display and saving

```
(gdb) c33 ts 0x20100 5 5
```

Saving PC trace information (`c33 tf` command)

The **`c33 tf`** command allows you to save the trace information currently saved in trace memory to a file. You also can specify the range of trace information to save by a cycle number. Otherwise, you can execute the `c33 ts` command to save only the searched information.

Example: Saves a range of trace information currently in trace memory that is equivalent to 10K clock cycles to the `trace.log` file

```
(gdb) c33 tf trace.log
```

Notes:
- Before trace information can be acquired using the `c33 td`, `c33 ts`, and `c33 tf` commands, memory map information must be set by the `c33 rpf` command. At this time, be sure to execute the `c33 rpf` command before the `target` command. If memory map information is set immediately before the program starts running, trace information cannot be acquired correctly.
- Because trace memory of the S5U1C33001H is large, it takes time for all trace data to be saved to a file. We recommend specifying a cycle number or setting search mode before saving.

  Example: To save trace information from cycle number 0 (latest) to 1000

  ```
  (gdb) c33 tf trace.log 1000 0
  ```

PC trace operation in ICD2/ICD3/ICD6 mode and precautions to be taken

Trace in ICD2/ICD3/ICD6 mode is performed in the manner described below.

The following four signals from the target CPU are connected to S5U1C33000H and S5U1C33001H inputs:

DST0, DST1, DST2:	These status signals indicate CPU execution status (i.e., whether executing successive instructions, a relative or absolute branch, or idle).
DPCO:	This signal serially outputs the PC value for the jump address when an absolute branch occurs.

This 4-bit information is saved for up to 128K clock periods in the S5U1C33000H, and up to 1M clock periods in the S5U1C33001H, synchronously with the CPU clock. Based on this information and related internal disassemble information, the debugger **gdb** performs the flow analysis below to obtain the PC value.

• DST0–2 = successive instruction execution:	One instruction is added.
• DST0–2 = relative branch:	Number of branch instructions is calculated from the disassemble information.
• DST0–2 = absolute branch:	Jump address is determined by referring to the DPCO information.

However, the trace starting point and corresponding PC value must be defined before analysis can be performed. Therefore, **gdb** uses the following methods to determine the PC value. "Method" indicates the method used as indicated by the displayed content of trace information.

Method: `SPC`	When the PC value at which execution started is known (all trace without overwrite)
Method: `DPC`	Determined from complete DPCO information obtained in an absolute branch
Method: `MAP`	Determined by supplementing map information for incomplete DPCO information obtained in an absolute branch Note that the absolute branch refers to a branch to the following instructions or interrupts: `call %rb, call.d %rb, jp %rb, jp.d %rb, ret, ret.d, reti, int`
Method: `TRG`	Determined by using a trigger address in an area trace
Method: `RET`	Determined by assuming that `call` and `ret` statements correspond one to one
Method: `RTI`	Determined by assuming that interrupt generated and `reti` correspond one for one

Due to the determination methods above, the following restrictions apply:

- Restrictions on trace with overwrite

  If trace is performed in a loop where a relative branch is repeated, for example, trace analysis cannot be performed until the PC value is determined. As a method to solve this problem, an 8-bit timer may be used to generate an interrupt every several ms (at which to output DPCO information), thereby making trace analysis possible. (See the Sample of `\gnu33\utility\sample_std\icdtrc`.)

- Restrictions on trace without overwrite

  If the program is started from a software PC breakpoint, **gdb** performs the following processing:
  1. Clear the software PC breakpoint at the address from which execution started.
  2. Single-step the first instruction.
  3. Set a software PC breakpoint back again.
  4. Execute subsequent instructions successively.

  The single-stepped parts of the program are not traced.

- Restrictions on area trace

  Two hardware PC breakpoints can be used during normal operation, but not in area trace mode because the debugger uses the breakpoints as a trace trigger. When the debugger is released from area trace, hardware PC breakpoints can be used again.

The following restrictions are common to all modes:

- Restrictions on an absolute branch

  If an absolute branch is repeated in less than 27 clock cycles, the PC value cannot be completely determined. Although recovery processing is attempted by using map information or a `call` to `ret` nest information, analysis may not always be possible.

- Restrictions on the executed program

  The debugger retains the debugging information it loaded with the `file` command and uses its disassembled information to perform analysis.

  Note that discrepancies may occur in the internal analysis information and target program in the following cases:
  - When the executed program exists in ROM
  - When the program dynamically copies and moves to execution routines
  - When the program area is changed by a debugger command, etc.

- Simulated I/O

  In simulated I/O, the debugger uses software PC breaks and single-stepping commands (including source-level stepping) internally. Therefore, simulated I/O cannot be used in combination with trace operation.

- Executed commands

  The trace data is sampled when the program is executed successively by the `continue` command, and not when the program is single-stepped. Also note that when the `continue` command is received, **gdb** deletes the previous trace data.

Precautions on the S5U1C33000H and S5U1C33001H hardware are as follows:

- Settings of the S5U1C33000H and S5U1C33001H

  To use the trace function in ICD2, ICD3 or ICD6 mode, set DIP switch SW4 of the S5U1C33000H or S5U1C33001H to the OPEN (up) position. Moreover, make sure the target board and S5U1C33000H or S5U1C33001H are connected using the 10-pin cable, with all signals required for trace (DST0–2 and DPCO) included.

- Upper-limit trace frequency of the S5U1C33000H and S5U1C33001H

  When tracing program execution with the S5U1C33000H or S5U1C33001H, the operating clock frequency is limited to 50 MHz. Above this clock frequency, garbled or illegible data may result.

  To run the CPU at a clock frequency higher than 50 MHz, use DIP SW4 to turn the trace function off. Also set the BCU (bus) speed of the S1C33 chip to 1/2 or less than that of the CPU core (by using the #X2SPD pin).

  All functions (except trace) operate at the same speed as the bus, the upper limit of which is 40 MHz.

Trace function in simulator mode

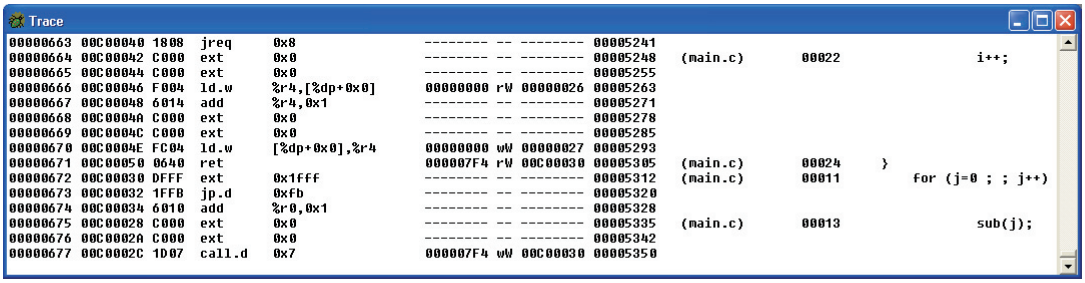
In simulator mode, you can use the **`c33 tm`** command to turn the trace function on or off, as well as specify the method of displaying data in a window or writing data to a file. When the trace function is turned on, trace results are displayed in a window or saved to a file for each instruction executed.

Example 1: To set a register-displayed trace with source information included and to specify file `trace.log` in which to save the information

```
(gdb) c33 tm on 4 trace.log
```

Example 2: To turn trace mode off

```
(gdb) c33 tm off
```



```
00000663 00C00040 1808 jreq    0x8                -------- -- -------- 00005241
00000664 00C00042 C000 ext     0x0                -------- -- -------- 00005248  (main.c)    00022                 i++;
00000665 00C00044 C000 ext     0x0                -------- -- -------- 00005255
00000666 00C00046 F004 ld.w    %r4,[%dp+0x0]      00000000 rW 00000026 00005263
00000667 00C00048 6014 add     %r4,0x1            -------- -- -------- 00005271
00000668 00C0004A C000 ext     0x0                -------- -- -------- 00005278
00000669 00C0004C C000 ext     0x0                -------- -- -------- 00005285
00000670 00C0004E FC04 ld.w    [%dp+0x0],%r4      00000000 wW 00000027 00005293
00000671 00C00050 0640 ret                        000007F4 rW 00C00030 00005305  (main.c)    00024    }
00000672 00C00030 DFFF ext     0x1fff             -------- -- -------- 00005312  (main.c)    00011          for (j=0 ; ; j++)
00000673 00C00032 1FFB jp.d    0xfb               -------- -- -------- 00005320
00000674 00C00034 6010 add     %r0,0x1            -------- -- -------- 00005328
00000675 00C00028 C000 ext     0x0                -------- -- -------- 00005335  (main.c)    00013                 sub(j);
00000676 00C0002A C000 ext     0x0                -------- -- -------- 00005342
00000677 00C0002C 1D07 call.d  0x7                000007F4 wW 00C00030 00005350
```

The trace information displayed in simulator mode is listed below.

<First line of each trace information>

Number Address Code Disassemble Address Type Data Clock [*File Line SourceCode*]

Number:	Number of executed instructions (in decimal) Number of instructions executed since the CPU was reset or trace turned on
Address:	Address of executed instructions (in hexadecimal)
Code:	Instruction codes (in hexadecimal)
Disassemble:	Disassembled contents of executed instructions
Address:	Accessed memory addresses (in hexadecimal)
Type:	Type of bus operation `rB`: Byte data read; `rH`: Halfword data read; `rW`: Word data read `wB`: Byte data write; `wH`: Halfword data write; `wW`: Word data write
Data:	Read/written data (in hexadecimal)
Clock:	Number of clock cycles executed (in decimal)
File:	Source file names (only when source display is selected by the `c33 tm` command)
Line:	Source line numbers (only when source display is selected by the `c33 tm` command)
SourceCode:	Source codes (only when source display is selected by the `c33 tm` command)

<Lines 2 to 4 (6) of each trace information>

These lines are displayed only when you choose to display register contents by using the `c33 tm` command. Register values are displayed in the following order:

For the C33 STD Core: `R0 R1 R2 R3 R4 R5 R6 R7`
`R8 R9 R10 R11 R12 R13 R14 R15`
`SP AHR ALR` `PSR` (Displayed individually for each flag.)

For the S1C33401: `R0 R1 R2 R3 R4 R5 R6 R7`
(C33 ADV Core) `R8 R9 R10 R11 R12 R13 R14 R15`
`SP AHR ALR` `PSR` (Displayed individually for each flag.)
`LCO LSA LEA SOR TTBR DP USP SSP`
`PSR` (Displayed individually for each flag added to the C33 ADV Core.)

This information is displayed in the [Trace] window when you choose to display in a window by using the `c33 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 (Clock) 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.

On chip bus trace (ICD3/ICD6 mode, C33 ADV Core only)

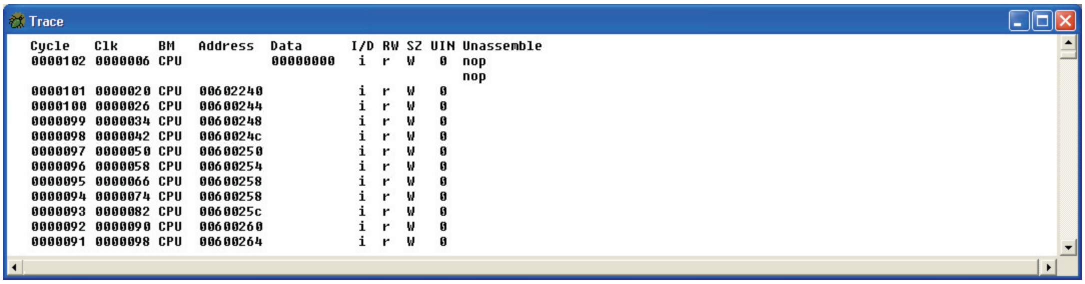
If the target incorporating the C33 ADV Core is to be debugged in ICD3/ICD6 mode, you can perform an on-chip bus trace using the internal facilities of the C33 ADV Core. The bus trace function acquires the bus cycle information and saves it in the trace memory when an access in the specified condition occurs on the specified bus.

By executing the **`c33 logiana`** command, bus trace information can be displayed/saved during debugging for the C33 STD or C33 PE Core model.

Bus trace information

The bus trace information is recorded in trace memory of the S5U1C33001H in the same way as for the PC trace information described above. Given below are examples of bus trace information displayed in the [Trace] window.

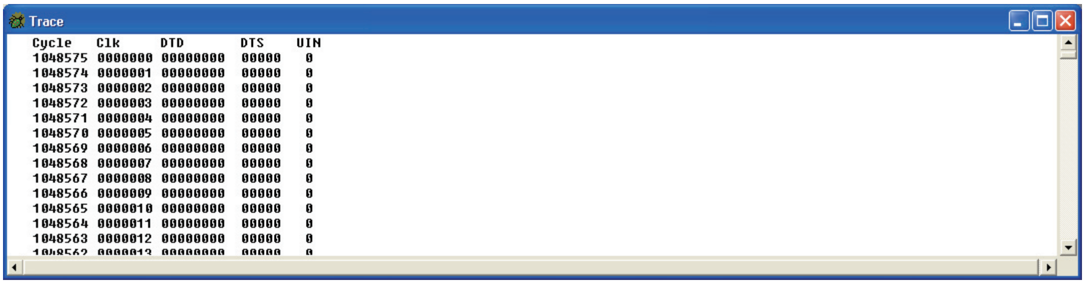
Example display in normal mode



```
Cycle   Clk      BM    Address   Data      I/D RW SZ UIN Unassemble
0000102 0000006 CPU             00000000   i  r  W   0  nop
                                                        nop
0000101 0000020 CPU   00602240             i  r  W   0
0000100 0000026 CPU   00600244             i  r  W   0
0000099 0000034 CPU   00600248             i  r  W   0
0000098 0000042 CPU   0060024c             i  r  W   0
0000097 0000050 CPU   00600250             i  r  W   0
0000096 0000058 CPU   00600254             i  r  W   0
0000095 0000066 CPU   00600258             i  r  W   0
0000094 0000074 CPU   00600258             i  r  W   0
0000093 0000082 CPU   0060025c             i  r  W   0
0000092 0000090 CPU   00600260             i  r  W   0
0000091 0000098 CPU   00600264             i  r  W   0
```

Cycle:	Trace cycles (decimal)	
	The information last written to trace memory assumes the value 000000.	
Clk:	Number of clock cycles executed	
	A cumulative number of clock cycles since the trace started is displayed.	
BM:	Bus type	
	`CPU`	Bus master = CPU (when HBCU selected)
	`CPU/CACHE`	Bus master = CPU or CACHE (when BBCU selected)
	`DMA/USR`	Bus master = DMA or USR (when BBCU selected)
Address:	Accessed address (hexadecimal)	
Data:	Read/written data (hexadecimal)	
I/D:	Data type	
	`i`	Instruction
	`d`	Data
RW:	Read/write	
	`r`	Read
	`w`	Write
SZ:	Access size	
	`B`	Byte
	`H`	Halfword
	`W`	Word
UIN:	USER_IN signal state	
	`0` or `1`	
Unassemble:	Disassembled instruction code	

Example display in user mode



```
Cycle   Clk     DTD       DTS    UIN
1048575 0000000 00000000  00000   0
1048574 0000001 00000000  00000   0
1048573 0000002 00000000  00000   0
1048572 0000003 00000000  00000   0
1048571 0000004 00000000  00000   0
1048570 0000005 00000000  00000   0
1048569 0000006 00000000  00000   0
1048568 0000007 00000000  00000   0
1048567 0000008 00000000  00000   0
1048566 0000009 00000000  00000   0
1048565 0000010 00000000  00000   0
1048564 0000011 00000000  00000   0
1048563 0000012 00000000  00000   0
```

Cycle: Trace cycles (decimal)
The information last written to trace memory assumes the value 000000.
Clk: Number of clock cycles executed
A cumulative number of clock cycles since the trace started is displayed.
DTD: Bus trace data (DTD pin output)
DTS: Bus trace data status (DTS pin output)
UIN: USER_IN signal state
`0` or `1`

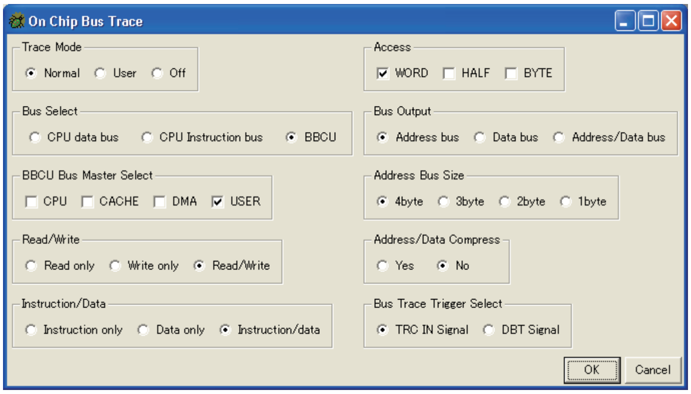
Although only the bus trace information is displayed in either mode in the above example, the debugger can also be set to display PC trace information.

Setting on chip bus trace (`c33 obt` command)

Use the **`c33 obt`** command or the [On Chip Bus Trace] dialog box to set bus trace conditions (select [Bus Trace] from the [View] menu to display the dialog).

Example: Set trace condition = word access by USER, address bus width = 4 bytes, and compressed output = no. Additionally, specify an external trace input signal (TRC IN signal) as the trace signal for user input (break trigger break).

```
(gdb) c33 obt 0 2 0x8 rw 2 0x04 0 4 0 0
Set bus trace
Trace mode                 : Normal mode
Bus select                 : BBCU
BBCU bus master select     : USER
Read/Write                 : Read/Write
Instruction/Data           : Instruction/Data
Access size                : Word
Bus output                 : Address bus
Address bus size           : 4 byte
Compression                : Disable
Bus trace trigger select   : TRC IN signal
```



Trace Mode: ● Normal ○ User ○ Off
Access: ☑ WORD ☐ HALF ☐ BYTE
Bus Select: ○ CPU data bus ○ CPU Instruction bus ● BBCU
Bus Output: ● Address bus ○ Data bus ○ Address/Data bus
BBCU Bus Master Select: ☐ CPU ☐ CACHE ☐ DMA ☑ USER
Address Bus Size: ● 4byte ○ 3byte ○ 2byte ○ 1byte
Read/Write: ○ Read only ○ Write only ● Read/Write
Address/Data Compress: ○ Yes ● No
Instruction/Data: ○ Instruction only ○ Data only ● Instruction/data
Bus Trace Trigger Select: ● TRC IN Signal ○ DBT Signal

For more information on these settings, refer to the section on the `c33 obt` command.

Displaying on chip bus trace information (`c33 otd` command)

The **`c33 otd`** command may be used to display in the [Trace] window only the bus trace information currently in trace memory or to display both PC trace and bus trace information. You also can specify the range of trace information to display by a cycle number.

Example 1: Displays a range of bus trace information currently in trace memory that is equivalent to 10K clock cycles.

```
(gdb) c33 otd 0
```

Example 2: Displays a range of PC trace and bus trace information currently in trace memory from the 1,000th to the latest cycle.

```
(gdb) c33 otd 1 1000 0
```

Saving on chip bus trace information (`c33 otf` command)

The **`c33 otf`** command may be used to save in a file only the bus trace information currently in trace memory or to save both PC trace and bus trace information. You also can specify the range of trace information to save by a cycle number.

Example 1: Save to a `trace.log` file a range of bus trace information currently in trace memory that is equivalent to 10K clock cycles.

```
(gdb) c33 otf 0 trace.log
```

Example 2: Save to the `trace.log` file a range of PC trace and bus trace information currently in trace memory from the 1,000th to the latest cycle.

```
(gdb) c33 otf 1 trace.log 1000 0
```

Time required to display trace data and create a file

Given below are examples of the approximate times needed to display trace data and create files, in conjunction with PC performance.

Output time per 10K Clk (10K Clk is the default size when a cycle range is not specified.)

Command	Pentium-3 450MHz	Pentium-4 3GHz
`c33 td`	10 sec	3 sec
`c33 otd 0`	13 sec	3 sec
`c33 otd 1`	23 sec	5 sec
`c33 tf`	2 sec	2 sec
`c33 otf 0`	1 sec	1 sec
`c33 otf 1`	2 sec	2 sec

Since displaying several hundreds to thousands of lines of information in the [Trace] window can be time-consuming, we recommend specifying a cycle range.

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 (`c33 stdin` command)

Set the following conditions in the **`c33 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) c33 stdin 1 READ_FLASH READ_BUF input.txt
```

Example 2: To turn the data input function off

```
(gdb) c33 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) c33 stdin 1 READ_FLASH READ_BUF
```

Output by stdout (`c33 stdout` command)

Set the following conditions in the **`c33 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) c33 stdout 1 WRITE_FLASH WRITE_BUF output.txt
```

Example 2: To turn the data output function off

```
(gdb) c33 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) c33 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 `c33 stdin` and `c33 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 `c33 stdin` and `c33 stdout` commands.

In the C/C++ source, define these labels in the lower-level `write` and `read` functions among the standard input/output library functions (see Section 7.4.4).

For examples of actual programs, refer to the sample programs and debugger command file installed in the `\gnu33\utility\sample_std\simio` 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 `c33 stdin` and `c33 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 `c33 stdin` or `c33 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 S5U1C33001H.

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 S1C33 chip or mounted on the target board. This utility and these commands can be used in the debugging environments of ICD2, ICD3, ICD6, and debug monitor modes.

Follow the procedure described below to write data to flash memory. For more details, refer to `readme.txt` for flash support utility **fls33g**. (Use `\gnu33\utility\fls33\fls33Vxxg.exe` to install **fls33g** and `readme.txt`.)

1. Loading flash routines

   Use the `load` command to load flash routines (erase and write routines) into internal RAM, etc.
   Example:

```
(gdb) file 29lv800t.elf              (Load debugging information.)
(gdb) target icd com1                (Connect the target.)
C33 ICD33 debugging
Connecting with target ... done
Target connection test ... done
Initializing ............ done
CPU cold resetting ....... done
ICD hardware version   ... 00
ICD software version   ... 2.0
Boot address ............ 0xC00000
0x00c00000 in ?? ()
(gdb) set {char}0x4812d=0x59         (Set boot vector address.)
(gdb) set {int}0x48134=0x600000
(gdb) load                           (Load flash routines.)
```

   Use these routines when actually erasing and writing data. For the routines provided by Seiko Epson, use addresses 0x30 to 0x7ff in internal RAM.

Note: Use the flash routines provided by Seiko Epson or those you have created yourself. The routines provided by Seiko Epson can be installed on your computer by executing `\gnu33\utility\fls33\fls33Vxxg.exe`. These routines are provided primarily for AMD flashes. Because source codes also are included, you can readily use the routines after making necessary corrections.

2. Setting flash memory (**`c33 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) c33 fls 0xc00000 0xcfffff FLASH_ERASE FLASH_LOAD
```

3. Erasing flash memory (**`c33 fle`** command)

   Erase the entire area or sectors of flash memory. The contents of flash memory will be changed to 0xff.

   Example: To erase all flash memory whose start address is 0xc00000

```
(gdb) c33 fle 0xc00000 0 0
```

   Always be sure to execute the `c33 fle` command after the `c33 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 `c33 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 S5U1C33001H (ICD3/ICD6 mode)

The S5U1C33001H incorporates a flash writer function, and **gdb** has commands to control this function.

To use the S5U1C33001H as a flash writer, set DSW1-1 to the ON (down) position. For details, refer to the "S5U1C33001H Manual (S1C33 Family In-Circuit Debugger)".

The flash writer control commands can only be used in ICD3/ICD6 mode, as described below.

Erasing programs/data (`c33 fwe` command)

The **`c33 fwe`** command erases the data erase/write program or write data and address information loaded in the S5U1C33001H.

Example 1: To erase write data

```
(gdb) c33 fwe 0
```

Example 2: To erase the data erase/write program

```
(gdb) c33 fwe 1
```

Loading a program (`c33 fwlp` command)

The **`c33 fwlp`** command loads the data erase/write program from the host in the S5U1C33001H 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) c33 fwlp writer.sa 0x90 0xb4
```

Loading data (`c33 fwld`, `c33 fwdc` commands)

The **`c33 fwld`** command loads the data to be written to flash memory from the host in the S5U1C33001H.

The **`c33 fwdc`** command loads the data saved in target board memory into the S5U1C33001H. 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) c33 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) c33 fwdc FLASH_START 0x100000 0 0 0
```

Displaying flash writer information (`c33 fwd` command)

The **`c33 fwd`** command displays information about the data loaded in the S5U1C33001H and information about the erase/write program.

```
(gdb) c33 fwd
CPU data address            : xxxxxxxx
Data size                   : xxxxxxxx
Erase start block           : xxxxxxxx
Erase end block             : xxxxxxxx
Erase parameter             : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

CPU program address         : xxxxxxxx
Program size                : xxxxxxxx
Erase routine entry address : xxxxxxxx
Write routine entry address : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

10.6.10 Support for Big Endian

The tools (C/C++ compiler to the linker) and libraries only support little endian. Note that the C/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 S1C33 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`, `c33 fg/fh/fw`, `c33 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, halfwords, or words according to the type of data in each command.

`load` command

Swaps data suitable for endian format and writes data in units of halfwords. For this reason, programs created by the C/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
			ICD2	ICD3	ICD6	SIM	MON	
Memory manipulation	**c33 fb**	Fill area (in bytes)	●	●	●	●	●	10-81
	c33 fh	Fill area (in half words)	●	●	●	●	●	10-81
	c33 fw	Fill area (in words)	●	●	●	●	●	10-81
	x /b	Memory dump (in bytes)	●	●	●	●	●	10-83
	x /h	Memory dump (in half words)	●	●	●	●	●	10-83
	x /w	Memory dump (in words)	●	●	●	●	●	10-84
	set {char}	Data input (in bytes)	●	●	●	●	●	10-85
	set {short}	Data input (in half words)	●	●	●	●	●	10-85
	set {int}	Data input (in words)	●	●	●	●	●	10-85
	c33 mvb	Copy area (in bytes)	●	●	●	●	●	10-86
	c33 mvh	Copy area (in half words)	●	●	●	●	●	10-86
	c33 mvw	Copy area (in words)	●	●	●	●	●	10-86
	c33 df	Save memory contents	●	●	●	●	●	10-88
	c33 rm	Read target memory	●	●	●	—	—	10-90
	c33 readmd	Memory read mode	—	●	●	—	—	10-91
Register manipulation	**info reg**	Display register	●	●	●	●	●	10-92
	set $	Modify register	●	●	●	●	●	10-95
Program execution	**continue**	Execute continuously	●	●	●	●	●	10-96
	until	Execute continuously with temporary break	●	●	●	●	●	10-98
	step	Single-step (every line)	●	●	●	●	●	10-100
	stepi	Single-step (every mnemonic)	●	●	●	●	●	10-100
	next	Single-step with skip (every line)	●	●	●	●	●	10-102
	nexti	Single-step with skip (every mnemonic)	●	●	●	●	●	10-102
	finish	Quit function	●	●	●	●	●	10-104
	c33 callmd	Set user function call mode	●	●	●	●	●	10-105
	c33 call	Call user function	●	●	●	●	●	10-106
CPU reset	**c33 rstc**	Cold reset (execute resetcold.gdb)	●	●	●	●	●	10-108
	c33 rsth	Hot reset (execute resethot.gdb)	●	●	●	●	●	10-110
	c33 rstt	Reset target	—	—	●	—	—	10-111
Interrupt	**c33 int**	Interrupt	—	—	—	●	—	10-112
Break	**break**	Set software PC break	●	●	●	●	●	10-113
	tbreak	Set temporary software PC break	●	●	●	●	●	10-113
	hbreak	Set hardware PC break	●	●	●	●	●	10-116
	thbreak	Set temporary hardware PC break	●	●	●	●	●	10-116
	watch	Set data write break	●	●	●	●	●	10-119
	rwatch	Set data read break	●	●	●	●	●	10-119
	awatch	Set data read/write break	●	●	●	●	●	10-119
	delete	Clear break by break number	●	●	●	●	●	10-121
	clear	Clear break by break position	●	●	●	●	●	10-123
	enable	Enable breakpoint	●	●	●	●	●	10-125
	disable	Disable breakpoint	●	●	●	●	●	10-125
	ignore	Disable breakpoint with ignore counts	●	●	●	●	●	10-127
	info breakpoints	Display breakpoint list	●	●	●	●	●	10-128
	c33 oab	Set on chip area (CE) break *1	—	●	●	—	—	10-129
	c33 obb	Set on chip bus break *1	—	●	●	—	—	10-131
	c33 timebrk	Set lapse of time break	—	—	●	—	—	10-135
	c33 savebreak	Save set breakpoints	●	●	●	●	●	10-136
	c33 loadbreak	Restore saved breakpoints	●	●	●	●	●	10-137
Symbol information	**info locals**	Display local symbol	●	●	●	●	●	10-138
	info var	Display global symbol	●	●	●	●	●	10-138
	print	Alter symbol value	●	●	●	●	●	10-139
File loading	**file**	Load debugging information	●	●	●	●	●	10-140
	load	Load program	●	●	●	●	●	10-141
Map information	**c33 rpf**	Set map information	●	●	●	●	●	10-142
	c33 map	Display map information	●	●	●	●	●	10-143
Flash memory manipulation	**c33 fls**	Set flash memory	●	●	●	—	●	10-144
	c33 fle	Erase flash memory	●	●	●	—	●	10-145
Trace	**c33 tm**	Set PC trace mode	●	●	●	●	—	10-146
	c33 td	Display PC trace contents	●	●	●	—	—	10-152
	c33 ts	Search PC trace contents	●	●	●	—	—	10-153
	c33 tf	Save PC trace contents	●	●	●	—	—	10-154
	c33 obt	Set on chip bus trace mode *1	—	●	●	—	—	10-155
	c33 otd	Display on chip bus trace contents *1	—	●	●	—	—	10-158
	c33 otf	Save on chip bus trace contents *1	—	●	●	—	—	10-160
Simulated I/O	**c33 stdin**	Data input simulation	●	●	●	●	●	10-161
	c33 stdout	Data output simulation	●	●	●	●	●	10-162

Classification	Command	Operation	Supported modes					Page
			ICD2	ICD3	ICD6	SIM	MON	
Flash writer	**c33 fwe**	Erase program/data	—	●	●	—	—	10-163
	c33 fwlp	Load program	—	●	●	—	—	10-164
	c33 fwld	Load data	—	●	●	—	—	10-165
	c33 fwdc	Copy target memory	—	●	●	—	—	10-166
	c33 fwd	Display flash writer information	—	●	●	—	—	10-167
Other	**c33 log**	Logging	●	●	●	●	●	10-168
	source	Execute command file	●	●	●	●	●	10-169
	c33 clockmd	Set execution counter mode	●	●	●	●	—	10-170
	c33 clock	Display execution counter	●	●	●	●	—	10-170
	target	Connect target	●	●	●	●	●	10-172
	detach	Disconnect target	●	●	●	●	●	10-174
	c33 das	Set debug unit address	—	●	●	—	—	10-175
	c33 lpt	Set/clear parallel loading mode	●	—	—	—	—	10-176
	pwd	Display current directory	●	●	●	●	●	10-177
	cd	Change current directory	●	●	●	●	●	10-177
	c33 firmupdate	Update ICD firmware	—	●	●	—	—	10-178
	c33 dclk	Change DCLK cycle	●	●	●	—	●	10-179
	c33 oscwait	Set OSC1 wait counter	—	●	●	—	—	10-180
	c33 cachehit	Display cache hit rate *1	—	●	●	—	—	10-181
	c33 logiana	Set logic analyzer mode	—	●	●	—	—	10-182
	c33 help	Help	●	●	●	●	●	10-183
	c33 savewatch	Store symbols registered in Watch Window	●	●	●	●	●	10-185
	c33 loadwatch	Register stored symbols to Watch Window	●	●	●	●	●	10-186
	quit	Quit debugger	●	●	●	●	●	10-187

Supported modes: ● = Can be used (same operation/display regardless of mode), ○ = Can be used (operation/display varies with mode),
— = Cannot be used

*1 Usable only when debugging the S1C33401 (C33 ADV Core)

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 ICD2, ICD3, ICD6, SIM, and MON in which the command can be used. You cannot use the command in modes other than those written here. When multiple instances of [ ] are entered, the command operates differently in each [ ].

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

c33 fb (fill area, in bytes)
c33 fh (fill area, in half words)
c33 fw (fill area, in words) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

`c33 fb` Rewrites specified memory area with specified byte data.
`c33 fh` Rewrites specified memory area with specified half word data.
`c33 fw` Rewrites specified memory area with specified word data.

Format

`c33 fb` ***StartAddr EndAddr Data***
`c33 fh` ***StartAddr EndAddr Data***
`c33 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 ≤ *StartAddr* ≤ *EndAddr* ≤ 0xffffffff, 0 ≤ *Data* ≤ 0xff (`c33 fb`), 0 ≤ *Data* ≤ 0xffff (`c33 fh`), 0 ≤ *Data* ≤ 0xffffffff (`c33 fw`)

Usage example

■ Example 1

```
(gdb) c33 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 data 0x01 (16 bytes).

■ Example 2

```
(gdb) c33 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 data 0x0001 (8 half words). (This applies to when using little endian.)

■ Example 3

```
(gdb) c33 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 data 0x00000001 (4 words). (This applies to when using little endian.)

Notes

- Writing in units of half words and words 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) c33 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 word boundaries, both are aligned to boundary addresses by setting the 2 low-order bits to 00 (LSB = 0 for half words). The following shows the actually executed command, where word addresses 0x0 to 0x8 (byte addresses 0x0 to 0xb) are rewritten with data 0x00000000.

  ```
  (gdb) c33 fw 0x0 0x8 0x0
  ```

- Address parameters are only effective for the 32 low-order bits, with excessive bits being ignored. For example, address 0x100000000 is processed as 0x00000000.
- Data parameters are only effective for the 8 low-order bits for `c33 fb`, 16 low-order bits for `c33 fh`, and 32 low-order bits for `c33 fw`, with excessive bits being ignored. For example, when data 0x100 is specified in `c33 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) [ICD2 / ICD3 / ICD6 / SIM / MON]

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 half words
`w` In units of words (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 ≤ *Length* ≤ 0xffffffff, 0 ≤ *Address* ≤ 0xffffffff

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 (8 half words or 4 words) 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.
- Address parameters are only effective for the 32 low-order bits, with excessive bits being ignored. For example, address 0x100000000 is processed as 0x00000000.
- Executing this command does not affect the [Memory] window.

set { } (data input) [ICD2 / ICD3 / ICD6 / SIM / MON]

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 half words
`int`	In units of words (default)

Address: Address to which to write data (decimal, hexadecimal, or symbol)

Data: The data to write (decimal, hexadecimal, or symbol)

Conditions: 0 ≤ *Address* ≤ 0xffffffff, 0 ≤ *Data* ≤ 0xff (`set {char}`), 0 ≤ *Data* ≤ 0xffff (`set {short}`), 0 ≤ *Data* ≤ 0xffffffff (`set {int}`)

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
```

Half word data 0x5555 is written to address 0x1000.

■ Example 3

```
(gdb) set {int}&i=0x55555555
(gdb) x /w &i
0x0 <i>: 0x55555555
```

Word data 0x55555555 is written to int variable `i`.

GUI

Memory contents can be changed in the [Memory] window. (See 10.4.5, "[Memory] Window".)

Notes

- Writing in units of half words and words 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.
- Address parameters are only effective for the 32 low-order bits, with excessive bits being ignored. For example, address 0x100000000 is processed as 0x00000000.
- Data parameters are only effective for the 8 low-order bits for `set {char}`, 16 low-order bits for `set {short}`, and 32 low-order bits for `set {int}`, 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.

c33 mvb (copy area, in bytes)
c33 mvh (copy area, in half words)
c33 mvw (copy area, in words) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

`c33 mvb`	Copies the content of a specified memory area to another area in units of bytes.
`c33 mvh`	Copies the content of a specified memory area to another area in units of half words.
`c33 mvw`	Copies the content of a specified memory area to another area in units of words.

Format

`c33 mvb` ***SourceStart SourceEnd Destination***
`c33 mvh` ***SourceStart SourceEnd Destination***
`c33 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 ≤ *SourceStart* ≤ *SourceEnd* ≤ 0xffffffff, 0 ≤ *Destination* ≤ 0xffffffff

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) c33 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) c33 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 int variable `i` is copied to an area located one word (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.
- Address parameters are only effective for the 32 low-order bits, with excessive bits being ignored. For example, address 0x100000000 is processed as 0x00000000.
- In `c33 mvh` and `c33 mvw`, addresses are adjusted to boundary addresses conforming to the data size. This is accomplished by processing the LSB address bit as 0 for `c33 mvh` and the 2 low-order address bits as 00 for `c33 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).

- 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 0xffffffff, data is only copied only up to 0xffffffff.
- 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.

c33 df (save memory contents) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Outputs the specified range of memory contents to a file in binary, text, or Motorola S3 format.

Format

`c33 df` ***StartAddr EndAddr Type Filename*** [***Append***]

StartAddr: Start address of area to save (decimal, hexadecimal, or symbol)
EndAddr: End address of area to save (decimal, hexadecimal, or symbol)
Type: One of the following values that specify the type of file
- `1` Binary file
- `2` Text file
- `3` Motorola S3 file

Filename: File name
Append:
- `a` Append mode enabled
  If *Type* = 1, dump data is appended to the end of a binary file when it is output.
  If *Type* = 2, dump data is appended to the end of a text file when it is output.
  If *Type* = 3, no footer records are appended to the end of a Motorola file.
- `f` Append mode enabled
  If *Type* = 1, dump data is appended to the end of a binary file when it is output.
  If *Type* = 2, dump data is appended to the end of a text file when it is output.
  If *Type* = 3, a footer record is appended to the end of a Motorola file.

If this specification is omitted, a new file is created.

Conditions: 0 ≤ *StartAddr* ≤ *EndAddr* ≤ 0xffffffff

Usage example

■ Example 1

```
(gdb) c33 df 0x0 0xf 2 dump.txt
Start address = 0x0, End address = 0xf, File type = Text
Processing 00000000-0000000F address.
```

Contents at addresses 0x0–0xf are written to file "`dump.txt`" in text format.
(Contents of `dump.txt`)

```
  addr   +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
00000000 00 01 02 03 04 05 06 07 00 01 02 03 04 0D 0E 0F
```

■ Example 2

```
(gdb) c33 df 0x80000 0x80103 3 dump.mot
Start address = 0x80000, End address = 0x8011f, File type = Motorola-S3
Processing 00080000-0008011F address.
```

Contents at addresses 0x80000–0x80103 are written to file "`dump.mot`" in Motorola S3 format. In Motorola S3 format, data is output 32 bytes per line. If the last line is less than 32 bytes, the end address is corrected so that the last line will be filled. In this example, specified end address 0x80103 is corrected to 0x8011f when writing data to a file.
(Contents of `dump.mot`)

```
S3250008000094D4BA020FCA086120800961881C6F0AA4D4BA020FCA086120800961881C6F0AA8
S3250008002008D3730A00000000000000000000000008D3730A09CA026139A505610CD309613F
                                                 :
S32500080100A4D5BA0278D5BA02A6D440004C000000C4D6BA0238E1260A8886730A2CD3280A32
S70500000000FA
```

■ Example 3

```
(gdb) c33 df 0x10 0x1f 2 dump.txt a
```

The contents of addresses 0x10–0x1f are appended in text form to the end of the file "`dump.txt`" when it is output.

■ Example 4

```
(gdb) c33 df 0x1000 0x1fff 3 dump.mot a ; Footer is not output. (First)
(gdb) c33 df 0x3000 0x3fff 3 dump.mot a ; Footer is not output.
(gdb) c33 df 0x5000 0x5fff 3 dump.mot a ; Footer is not output.
(gdb) c33 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

- Address parameters are only effective for the 32 low-order bits, with excessive bits being ignored. For example, address 0x100000000 is processed as 0x00000000.
- If the end address is smaller than the start address, an error is assumed.

c33 rm (read target memory) [ICD2 / ICD3 / ICD6]

Operation

Reads the program stored in the target memory into the debugger. The read data is used to analyze PC trace information by the ICD (S5U1C33000H/S5U1C33001H). This command should be used when the program on the target board is rewritten using a memory manipulation command or when acquiring PC trace information by executing the program after transferring it to the execution area.

Format

`c33 rm` ***Address1 Address2***

Address1: Start address of area to be read (decimal or hexadecimal)
Address2: End address of area to be read (decimal or hexadecimal)
Conditions: 0 ≤ *Address1* ≤ *Address2* ≤ 0xffffffff

Usage example

```
(gdb) c33 rm 0 0x7fe
```

The contents of the target board memory addresses 0x0 to 0x7fe are read into the debugger.

Notes

- The `c33 rm` command is effective only in ICD2, ICD3, or ICD6 mode.
- The address range to be read using this command must be configured as a valid memory area by loading a parameter file.
- When the program is executed immediately after it is transferred to the execution area, correct PC trace information (operation codes, disassemble information) cannot be acquired.
  Before executing the program at the destination address, the destination memory information (program code) must be read into the debugger.
  Examples:
  1) When the program stored in addresses 0xc0100 to 0xc01fff are copied to addresses 0x1000 to 0x1fff of the internal RAM and then executed from address 0x1000:
     In this case, the command shown below must be executed before starting the program copied into the internal RAM from address 0x1000.
     ```
     (gdb) c33 rm 0x1000 0x1fff
     ```
  2) When acquiring the PC trace information before and after transferring the program code:
     In this case, copy the program to the destination using a debug command and then read the transferred program information into the debugger using this command as a preparation. After that execute a normal sequence (executing the program → stop the execution using a break function → acquiring the trace information).
     ```
     (gdb) c33 mvh 0xc01000 0xc01fff 0x1000     Copy the program to the destination
     (gdb) c33 rm 0x1000 0x1fff                 Read the transferred program into the debugger
     (gdb) c33 tm 1 1                           Set up the trace mode (if necessary)
     (gdb) c33 rsth
     (gdb) cont                                 Execute the program
                        :
                        :                       ← Break
     (gdb) c33 tf trace.txt                     Save the PC trace information to a file
     ```

c33 readmd (memory read mode) [ICD3 / ICD6]

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`, `ld.h`, or `ld.w` instruction is used to access memory in units of bytes, halfword (two-byte) units, or word (four-byte) units, respectively.

Use boundary exact mode if reading data in units of bytes every time becomes inconvenient.

Format

`c33 readmd` ***Mode***

Mode: Selects memory read mode
`0` Normal mode (default)
`1` Boundary exact mode

Usage example

```
(gdb) c33 readmd 1
```

Selects boundary exact mode.

Notes

- The `c33 readmd` command is effective only in ICD3 or ICD6 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.
  `c33 df, x`

10.7.4 Register Manipulation Commands

info reg (display register) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Displays the contents of the CPU registers.

Format

`info reg` [*RegisterName*]

RegisterName: Name of register to display (specified in lowercase letters)
`r0-r15`, `psr`, `sp`, `alr`, `ahr`, `lco`, `lsa`, `lea`, `sor`, `ttbr`, `dp`, `idir`, `dbbr`, `usp`, `ssp`, `pc`
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             0xaaaaaaaa  -1431655766
(gdb) info reg ttbr
ttbr           0x20000000  536870912
```

When a register name is specified, only the content of that register is displayed.

■ Example 2 (If the target is the C33 STD Core)

```
(gdb) info reg
r0             0xd20       3360
r1             0xaaaaaaaa  -1431655766
r2             0xaaaaaaaa  -1431655766
r3             0xaaaaaaaa  -1431655766
r4             0x690       1680
r5             0xaaaaaaaa  -1431655766
r6             0x0         0
r7             0xaaaaaaaa  -1431655766
r8             0xaaaaaaaa  -1431655766
r9             0xaaaaaaaa  -1431655766
r10            0xaaaaaaaa  -1431655766
r11            0xaaaaaaaa  -1431655766
r12            0xaaaaaaaa  -1431655766
r13            0xaaaaaaaa  -1431655766
r14            0xaaaaaaaa  -1431655766
r15            0x0         0
psr            0x2         2
sp             0x7f8       2040
alr            0x0         0
ahr            0x0         0
pc             0xc00030    12582960
```

■ Example 3 (If the target is the S1C33401 (C33 ADV Core))

```
(gdb) info reg
r0              0xd20       3360
r1              0xaaaaaaaa  -1431655766
r2              0xaaaaaaaa  -1431655766
r3              0xaaaaaaaa  -1431655766
r4              0x690       1680
r5              0xaaaaaaaa  -1431655766
r6              0x0         0
r7              0xaaaaaaaa  -1431655766
r8              0xaaaaaaaa  -1431655766
r9              0xaaaaaaaa  -1431655766
r10             0xaaaaaaaa  -1431655766
r11             0xaaaaaaaa  -1431655766
r12             0xaaaaaaaa  -1431655766
r13             0xaaaaaaaa  -1431655766
r14             0xaaaaaaaa  -1431655766
r15             0x0         0
psr             0x2         2
sp              0x7f8       2040
alr             0x0         0
ahr             0x0         0
lco             0x0         0
lsa             0x0         0
lea             0x0         0
sor             0x0         0
ttbr            0x20000000  536870912
dp              0x0         0
idir            0x40000000  67108864
dbbr            0x60000     393216
usp             0x0         0
ssp             0x0         0
pc              0xc00030    12582960
```

■ Example 4 (If the target is the C33 PE Core)

```
(gdb) info reg
r0              0xd20       3360
r1              0xaaaaaaaa  -1431655766
r2              0xaaaaaaaa  -1431655766
r3              0xaaaaaaaa  -1431655766
r4              0x690       1680
r5              0xaaaaaaaa  -1431655766
r6              0x0         0
r7              0xaaaaaaaa  -1431655766
r8              0xaaaaaaaa  -1431655766
r9              0xaaaaaaaa  -1431655766
r10             0xaaaaaaaa  -1431655766
r11             0xaaaaaaaa  -1431655766
r12             0xaaaaaaaa  -1431655766
r13             0xaaaaaaaa  -1431655766
r14             0xaaaaaaaa  -1431655766
r15             0x0         0
psr             0x2         2
sp              0x7f8       2040
alr             0x0         0
ahr             0x0         0
ttbr            0x20000000  536870912
idir            0x600000000 1610612736
dbbr            0x60000     393216
pc              0xc00030    12582960
```

GUI

The contents of all registers can be confirmed in the [Registers] window. (See 10.4.4, "[Registers] Window".)

Notes

- This command will display registers only those contained in the currently targeted C33 Core.
- 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) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Changes the values of the CPU registers.

Format

`set $`*RegisterName*`=`*Value*

RegisterName: Name of register to change (specified in lowercase letters)
`r0-r15`, `psr`, `sp`, `alr`, `ahr`, `lco`, `lsa`, `lea`, `sor`, `ttbr`, `dp`, `usp`, `ssp`, `pc`
Value: Word data to set in the register (decimal, hexadecimal, or symbol)
Conditions: 0 ≤ *Value* ≤ 0xffffffff

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

- The set values are only effective for the 32 low-order bits, with excessive bits being ignored. For example, value 0x100000000 is processed as 0x00000000.
- 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) [ICD2 / ICD3 / ICD6 / SIM / MON]

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 (except in debug monitor mode).

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 `c33 rsth` (hot 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.

until (execute continuously with temporary break) [ICD2 / ICD3 / ICD6 / SIM / MON]

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/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 (except in debug monitor mode).
- 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 ≤ address ≤ 0xffffffff

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 `c33 rsth` (hot reset) before the `until` command.
- If the location set as a temporary breakpoint is a C/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, the address parameter is only effective for the 32 low-order bits, with excessive bits being ignored. For example, address 0x100000000 is processed as 0x00000000.
- 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) [ICD2 / ICD3 / ICD6 / SIM / MON]

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/C++ sources, one line of C/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 (except in debug monitor mode).

Format

`step` `[`*Count*`]`
`stepi` `[`*Count*`]`

Count: Number of steps to execute (decimal or hexadecimal)
One step is assumed if omitted.
Conditions: 1 ≤ *Count* ≤ 0x7fffffff

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 `c33 rsth` (hot 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. Note that `halt` and `slp`-actuated standby mode is not supported in simulator mode.

next (single-step with skip, every line)
nexti (single-step with skip, every mnemonic) [ICD2 / ICD3 / ICD6 / SIM / MON]

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/C++ sources, one line of C/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 (except in debug monitor mode).

Format

`next` [*Count*]
`nexti` [*Count*]

Count:	Number of steps to execute (decimal or hexadecimal) One step is assumed if omitted.
Conditions:	$1 \le Count \le$ 0x7fffffff

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 `c33 rsth` (hot 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. Note that `halt` and `slp`-actuated standby mode is not supported in simulator mode.

finish (finish function) [ICD2 / ICD3 / ICD6 / SIM / MON]

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 (except in debug monitor mode).

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.

c33 callmd (set user function call mode) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Sets the destination at which to output execution results after executing the `c33 call` (user function call) command.

Format

`c33 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) c33 callmd 2 call.txt
```

The execution results of the `c33 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 `c33 call` command execution, with the results written to the file (which is then closed) when the `c33 call` command is completed. If an existing file name is specified, the file is overwritten with the new execution results.

c33 call (call user function) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Calls a user function.
However, an assembler entry program is required.

Format

`c33 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) c33 callmd 1
(gdb) c33 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 `c33 call` is invoked, **gdb** executes a specified user function and receives a return value from `%r10`. If the return value is -1 (0xffffffff), **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 `c33 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 word (4-byte) boundary. A user function must set the start address of this packet in `%r10`. If packets cannot be returned, the user function should set -1 in `%r10`.
The following shows an example of a user function to be called.
The user function must always end with "`jp %r15`".

Example entry program (assembler)

```
#define  SP_INI      0x0800      ; sp is in end of 1KB internal RAM
#define  DP_INI      0x0000      ; default data area pointer dp (%r15) is 0x0

         .txt
.global print_data
print_data:
         ld.w  %r5,%sp           ; save SP
         xld.w %r4,SP_INI
         ld.w  %sp,%r4           ; set SP
         ld.w  %r4,%r15          ; save return address
         pushn %r8               ; save registers
         ld.w  %r6,%r12
         ld.w  %r7,%r13
         ld.w  %r8,%r14
         xld.w %r15,DP_INI       ; set default data area pointer for safety
         xcall iprint_data       ; enter C program
         ld.w  %r10,%r4
         popn  %r8               ; restore registers
         ld.w  %sp,%r5           ; restore SP
         ld.w  %r15,%r4          ; restore return address
         jp    %r15              ; back to mini monitor
```

User function

```
struct {
        int  size;
        char buf[0x100];
} tmpbuf;

int *iprint_data(int arg1, int arg2, int arg3)
{
        int cnt;
        cnt = sprintf(tmpbuf.buf, "arg1=%d, arg2=%d, arg3=%d", arg1, arg2, arg3);
        tmpbuf.size = cnt+1;          // +1 is null
        return (int*)&tmpbuf;
}
```

Notes

- If any function without an entry program is called, **gdb** may run uncontrollably.
- Before executing the `c33 call` command, be sure to delete or disable all break settings in the called function. If any break is set in it, **gdb** may not operate normally.

10.7.6 CPU Reset Commands

c33 rstc (cold reset) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Cold-resets the CPU.
As a result, the CPU is reset to its initial state as shown below.

(1) Internal registers of the CPU

`R0-R15:` `0xaaaaaaaa`
`PC:` Boot address (indicated by content of 0xc00000)
`SP:` `0x0aaaaaa8` (C33 STD, C33 PE)
`0x00000000` (C33 ADV)
`PSR:` `0x00000000`
`AHR, ALR:` `0xaaaaaaaa`
`LCO:` `0x00000000` (C33 ADV)
`LSA:` `0x00000000` (C33 ADV)
`LEA:` `0x00000000` (C33 ADV)
`SOR:` `0x00000000` (C33 ADV)
`TTBR:` `0x20000000` (C33 ADV, C33 PE)
`DP:` `0x00000000` (C33 ADV)
`IDIR:` `0x40000000` (C33 ADV)
`0x60000000` (C33 PE)
`DBBR:` `0x00060000` (C33 ADV, C33 PE)
`USP:` `0x00000000` (C33 ADV)
`SSP:` `0x00000000` (C33 ADV)

The registers are shared by all cores unless otherwise noted.

(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 redisplays the program beginning with that address. The [Registers] window reappears with the same settings as (1).

Format

```
c33 rstc
```

Usage example

```
(gdb) c33 rstc
```

The CPU is cold-reset.

GUI

The `c33 rstc` command can also be executed using one of the methods below.

- Click the [Reset cold] button in the [Source] window.
  [Reset cold] button
- Choose [Reset cold] from the [Reset] menu in the [Source] window.

The above menu command/button executes the command file `\gnu33\resetcold.gdb`.
Contents of `resetcold.gdb` at shipment

```
c33 rstc
```

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 other than simulator mode, refer to the technical manual provided with your MCU for details on how to initialize buses and I/Os.
- In ICD2, ICD3, and ICD6 modes, the processing described above is performed, with the S1C33 chip also being reset. The target board is not reset. If the target runs in free-running mode when the `c33 rstc` command is executed, reset the target after applying a forcible break. When the target connected to the S5U1C33000H or S5U1C33001H is reset, the target system enters free-running mode, and can be made to stop running by using the `c33 rstc` command.
- When debugging the C33 ADV Core in simulator mode and no map is set for one byte or more from address 0x20000000 in a parameter file, the PC value is initialized to the address indicated by the content of 0xc00000.

c33 rsth (hot reset) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Hot-resets the CPU.
All internal registers (except the PC register) and the execution counter are initialized and the windows reappear the same way as with the `c33 rstc` command. The PC value (boot address) is specified by TTBR.

Format

```
c33 rsth
```

Usage example

```
(gdb) c33 rsth
```

The CPU is hot-reset.

GUI

The `c33 rsth` command can also be executed using one of the methods below.

- Click the [Reset hot] button in the [Source] window.

  [Reset hot] button
- Choose [Reset hot] from the [Reset] menu in the [Source] window.

The above menu command/button executes the command file `\gnu33\resethot.gdb`.

Contents of `resethot.gdb` at shipment

```
c33 rsth
```

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 other than simulator mode, the bus status and I/O status are retained.
- When the following conditions are true when operating in simulator mode, the address indicated by the content of 0xc00000 is set on the PC.
  - Simulator mode
  - C33 ADV mode
  - TTBR = 0x20000000
  - One byte or more from address 0x20000000 in the loaded parameter file are mapped to ROM or RAM.

c33 rstt (reset target) [ICD6]

Operation

Outputs a reset signal from the S5U1C33001H1400 (ICD Ver.6) to the reset input pin on the target board.

Format

```
c33 rstt
```

Usage example

```
(gdb) c33 rstt
TARGET resetting ................... done
```

The target is reset.

Notes

- The `c33 rstt` command can only be used in ICD6 mode.
- To execute this command, a reset input pin is required on the target board.
- The following message is displayed if the target cannot be reset:

```
TARGET resetting ................... failure
```

10.7.7 Interrupt Command

c33 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

```
c33 int [Type Level]
```

Type: Interrupt type (decimal, hexadecimal, or symbol)
Level: Interrupt priority level (decimal, hexadecimal, or symbol)
Conditions: $0 \leq Type \leq$ 0x7fff, $0 \leq Level \leq 15$

Usage example

■ Example 1

```
(gdb) c33 int
```

If no parameters are specified, an NMI is generated.

■ Example 2

```
(gdb) c33 int 3 6
```

Any maskable interrupt number and its priority level can be set.

Notes

- The `c33 int` command can only be used in simulator mode.
- Make sure the interrupt type is specified from 0 to 0x7fff. If this range is exceeded, an error is assumed.
- Make sure the interrupt priority level is specified from 0 to 15. If this range is exceeded, an error is assumed.
- The boot vector address (TTBR) setting is effective even in simulator mode.

10.7.8 Break Setup Commands

break (set software PC break)
tbreak (set temporary software PC break) [ICD2 / ICD3 / ICD6 / SIM / MON]

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: $0 \le$ address $\le$ 0xffffffff

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. (Breakpoints are effective when set and remain effective unless disabled.)
`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 (including temporary breaks) 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/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/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/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/C++ source (i.e., instruction to be expanded to mnemonic) in the function. Although a `push` 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.
- If software PC breakpoints are specified with an address value, the address parameter is only effective for the 32 low-order bits, with excessive bits being ignored. For example, address 0x100000000 is processed as 0x00000000.
- When specifying software 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.
- 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.
- No software PC breaks and temporary software PC breaks can be set in areas assigned the "ROM" attribute in a parameter file (`*.par`).

hbreak (set hardware PC break)
thbreak (set temporary hardware PC break) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Sets a hardware PC breakpoint. This breakpoint can be set at up to two locations. 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 ≤ address ≤ 0xffffffff

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. (Breakpoints are effective when set and remain effective unless disabled.)
`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".

Notes

- Hardware PC breakpoints (including temporary breaks) can be set at up to two locations. If this limit is exceeded, an error is assumed. Note that this break count includes the hardware PC breakpoints used by the debugger as it runs the program with a temporary break attached.
- Area traces in ICD2, ICD3 and ICD6 modes use the hardware PC break facility. When you set area trace mode, the hardware PC break facility is disabled and becomes unusable. However, hardware PC breakpoints are not cleared when area trace mode is reset (and full trace mode set), but are enabled again.
- The temporary hardware PC breakpoints you set are not displayed in the [Breakpoints] and [Source] windows.
- C/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/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/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/C++ source (i.e., instruction to be expanded to mnemonic) in the function. Although a `push` 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.
- If hardware PC breakpoints are specified with an address value, the address parameter is only effective for the 32 low-order bits, with excessive bits being ignored. For example, address 0x100000000 is processed as 0x00000000.
- 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.

watch (set data write break)
rwatch (set data read break)
awatch (set data read/write break) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Sets a data breakpoint. This breakpoint can be set at only one location.

When you run the target program after setting a data break, the program breaks immediately after accessing the specified address. A symbol or address can be specified for the breakpoint.

The primary difference between the three commands described here is the method of access to allow the program to break.

`watch`: When a data break is set by the `watch` command, the target program is made to break when writing data to a specified address.

`rwatch`: When a data break is set by the `rwatch` command, the target program is made to break when reading data from a specified address.

`awatch`: When a data break is set by the `awatch` command, the target program is made to break when writing data to or reading data from a specified address.

Format

`watch` ***Breakpoint***
`rwatch` ***Breakpoint***
`awatch` ***Breakpoint***

Breakpoint: Breakpoint

A breakpoint can be specified by either of the following:

- Symbol

Conditions:
- Must not be double type, structure, union name, or array name, unless it is a member of a structure/union or attached to an array index.
- *Address (decimal, hexadecimal, or symbol)

Conditions: 0 ≤ Address ≤ 0xffffffff

Usage example

■ Example 1

```
(gdb) watch i
Hardware watchpoint 1: i
(gdb) continue
Continuing.
Hardware watchpoint 1: i

Old value = 0
New value = 1
sub (k=1) at ./main.c:24
```

A data write breakpoint is set at the address of variable `i`.

The program being executed is made to break when data is written to variable `i`.

■ Example 2

```
(gdb) rwatch i
Hardware read watchpoint 2: i
(gdb) continue
Continuing.
Hardware read watchpoint 2: i

Value = 2
0x00c00042 in sub (k=1) at ./main.c:22
```

A data read breakpoint is set at the address of variable `i`.

The program being executed is made to break when data is read from variable `i`.

■ Example 3

```
(gdb) awatch *0x0
Hardware access (read/write) watchpoint 3: *0
(gdb) continue
Continuing.
Hardware access (read/write) watchpoint 3: *0

Value = 2
sub (k=1) at ./main.c:24
```

A data read/write breakpoint is set at address 0x0.
The program being executed is made to break when data is read from or written to address 0x0.

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. (Breakpoints are effective when set and remain effective unless disabled.)
`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

You can also use the following methods to set data breakpoints.

1. Place the [Source] window into SOURCE display mode to display the source file.
2. Place the cursor on a global variable name you want to set as the breakpoint.
3. Right-click to display a popup menu and select one from the following menu commands.

[Set Read Data Breakpoint at <*Symbol*>]	Sets a data read break
[Set Write Data Breakpoint at <*Symbol*>]	Sets a data write break
[Set Read/Write Data Breakpoint at <*Symbol*>]	Sets a data read/write break

To clear the breakpoint in the same way, right-click on the global variable name set as the breakpoint to display a popup menu and select [Delete Data Breakpoint at <*symbol*>] from the menu.

Notes

- Data breakpoints can be set at only one location. If this limit is exceeded, an error is assumed.
- The data breakpoints you set are not displayed in the [Breakpoints] and other windows.
- If data breakpoints are specified using a nonexistent symbol, an error is assumed.
- If data breakpoints are specified with an address value, the address parameter is only effective for the 32 low-order bits, with excessive bits being ignored. For example, address 0x100000000 is processed as 0x00000000.
- Unless the location specified for a data breakpoint is the boundary address conforming to the data size to which the target program will access, the program will not break. For example, to make the program break upon accessing an int variable allocated to address 0x0, the program will not break if the address you specify is between 0x1 to 0x3. Make sure the breakpoint address you specify has its two low-order bits set to 00 when accessing in units of words (32 bits), or its least significant bit set to 0 when accessing in units of halfwords (16 bits).
- The program stops at one to several instructions after break conditions are met.

delete (clear break by break number) [ICD2 / ICD3 / ICD6 / SIM / MON]

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   hw watchpoint  keep y              i
```

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   hw watchpoint  keep y              i
```

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 and data 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/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.
- Breaks set by the `c33 oab` or `c33 obb` command cannot be cleared by the `delete` command.

clear (clear break by break position) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Deletes PC breakpoints currently set individually by specifying a set position (function name, line number, or address). This command is used to delete software PC breakpoints (including temporary software PC breakpoints) and hardware PC breakpoints, but cannot be used to delete data breakpoints.

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: $0 \le \text{address} \le 0\text{xffffffff}$

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
5   hw watchpoint  keep y              i
```

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
5   hw watchpoint  keep y              i
```

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
5   hw watchpoint  keep y              i
```

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
Num Type           Disp Enb Address    What
5   hw watchpoint  keep y              i
```

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.

■ Example 4

```
(gdb) clear *i
No breakpoint at *i.
(gdb) info breakpoints
Num Type           Disp Enb Address    What
5   hw watchpoint  keep y              i
```

The data breakpoint set at variable `i` cannot be deleted by the `clear` command.

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 and data 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/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 cannot be used to delete data breakpoints. Use the `delete` command instead.
- 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.
- Breaks set by the `c33 oab` or `c33 obb` command cannot be cleared by the `clear` command.

enable (enable breakpoint)
disable (disable breakpoint) [ICD2 / ICD3 / ICD6 / SIM / MON]

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
5   hw watchpoint  keep y              i
```

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
5   hw watchpoint  keep y              i
```

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) disable 5
(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
5   hw watchpoint  keep n              i
```

Data breakpoints can also be switched between enabled and disabled states.

■ Example 3

```
(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
5   hw watchpoint  keep y              i
```

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 and data 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/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.
- Breaks set by the `c33 oab` or `c33 obb` command cannot be enabled/disabled by the `enable/disable` command.

ignore (disable breakpoint with ignore counts) [ICD2 / ICD3 / ICD6 / SIM / MON]

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.
- Breaks set by the `c33 oab` or `c33 obb` command cannot be disabled by the `ignore` command.

info breakpoints (display breakpoint list) [ICD2 / ICD3 / ICD6 / SIM / MON]

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
3   hw watchpoint  keep y              i
  breakpoint already hit 6 times
```

Num: Indicates a break number.

Type: Indicates the type of breakpoint.

`breakpoint`	Software PC breakpoint
`hw breakpoint`	Hardware PC breakpoint
`hw watchpoint`	Data 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). This information is not displayed for data breaks.

What: Indicates the location where a breakpoint is set. For PC breakpoints, this information is displayed in "`in` *function name* `at` *source file name*:*line number*" format. For data breakpoints, specified symbols or addresses are displayed.

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".

Notes

Break informations set by the `c33 oab` or `c33 obb` command cannot be displayed by the `info breakpoints` command.

c33 oab (set on-chip area break) [ICD3 / ICD6]

Operation

Configures the area break function that breaks program execution according to the bus master, accessed area, and read/write conditions.

This command is effective only when the target processor incorporates the C33 ADV Core.

The area break status is displayed on the right of the display mode combo box.

When the area break is enabled	area: on
When the area break is disabled	area: off

Format

`c33 oab` ***Mode BBCU_BusMaster R/W Area***

Mode: On-chip area break mode

`on` Sets on-chip area break

`off` Clears on-chip area break

When clearing area break, do not specify the *BBCU_BusMaster* and the following parameters.

BBCU_BusMaster: BBCU bus master (decimal or hexadecimal)

Specify a value from 0x01 to 0x0f with the bit to be selected set to 1.

(One or more conditions are selectable. At least one condition must be selected.)

bit 3 = 1: Memory access by the USER (CPU)
bit 2 = 1: Memory access by the DMA
bit 1 = 1: Memory access for refill/write back by the Cache
bit 0 = 1: Memory access by the CPU

R/W: Read/Write condition

`r` Read

`w` Write

`rw` Read/Write

Area: Area accessed (decimal or hexadecimal)

Areas 0 to 22 can be specified with a 23-bit value from 0x000001 to 0x7fffff. Bit 0 represents Area 0 and Bit 22 represents Area 22. To include areas in the break condition, set the corresponding bits to 1. To remove areas from the break condition, set the corresponding bits to 0.

Usage example

■ Example 1

```
(gdb) c33 oab on 0x05 w 0x3e2070
Area break on
BBCU bus master select :DMA,CPU
Read/Write             :Write
Area                   :4,5,6,13,17,18,19,20,21
```

The program being executed is made to break when the DMA or HBCU writes data to an area within Areas 4–6, 13, and 17–21.

When this break occurs, the following message is output:

```
Hardware on chip Area Break!
<<<Area Break Status>>>
    Bus master : HBCU(CPU)          ;Bus master    HBCU(CPU)/CCU/DMA/USER
    Access     : Write              ;Access        Read/Write
    Area       : 4                  ;Area number   (0–22)
    Data type  : Data               ;Data type     Instruction/Data
    Data size  : Word               ;Data size     Byte/Half Word/Word
```

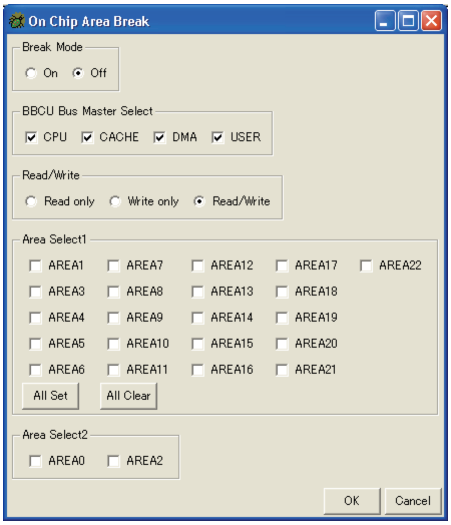
■ Example 2

```
(gdb) c33 oab off
Area break off
```

The on-chip area break conditions are cleared.

GUI

The area break conditions can be set in the [On Chip Area Break] dialog box. Select [Area Break] from the [View] menu to display the dialog box.



On Chip Area Break

Break Mode: ○ On ● Off

BBCU Bus Master Select: ☑ CPU ☑ CACHE ☑ DMA ☑ USER

Read/Write: ○ Read only ○ Write only ● Read/Write

Area Select1:
☐ AREA1 ☐ AREA7 ☐ AREA12 ☐ AREA17 ☐ AREA22
☐ AREA3 ☐ AREA8 ☐ AREA13 ☐ AREA18
☐ AREA4 ☐ AREA9 ☐ AREA14 ☐ AREA19
☐ AREA5 ☐ AREA10 ☐ AREA15 ☐ AREA20
☐ AREA6 ☐ AREA11 ☐ AREA16 ☐ AREA21

Area Select2: ☐ AREA0 ☐ AREA2

Break Mode:	Select [On] to set and enable on-chip area break conditions or select [Off] to disable on-chip area break.
BBCU Bus Master Select:	Select one or more BBCU bus masters.
Read/Write:	Select the read/write condition.
Area Select1/Area Select2:	Select one or more areas to break program execution when the area is accessed.
[All Set] button:	Use this button to select all check boxes in the [Area Select1] field.
[All Clear] button:	Use this button to deselect all check boxes in the [Area Select1] field.
[OK] button:	When this button is clicked, the debugger sends the command with the values set here to ICD and closes the dialog box. If there is no selected check box in the [BBCU Bus Master Select] or [Area Select1/Area Select2] field, a message box with an error message is displayed.
[Cancel] button:	When this button is clicked, the debugger discards the above settings and closes the dialog box.

When the dialog box is reopened, it shows the previous set status. The first time the dialog box opens, it shows the default settings as the screen sample above.

Notes

- The `c33 oab` command is effective only in ICD3 and ICD6 modes.
- This command does not support the Cache in Area 2 and the MMU area.
- A break in Area 0 or Area 3 occurs only when the MMU translates the accessed address into an address in Area 0 or Area 3.
- Note that an error occurs if the target processor does not support the on-chip area break function.

c33 obb (set on-chip bus break) [ICD3 / ICD6]

Operation

Configures the bus break function that breaks program execution according to the bus status and other various conditions.

This command is effective only when the target processor incorporates the C33 ADV Core.

The bus break status is displayed on the right of the display mode combo box.

When the bus break is enabled	bus: on
When the bus break is disabled	bus: off

Format

`c33 obb` *Mode Bus BBCU_BusMaster Trig/Break R/W Inst/Data Access Int IntLevel DataComp Address AddressMask Data DataMask Counter CounterClr/Save*

Mode: On-chip bus break mode

`on` Sets on-chip bus break

`off` Clears on-chip bus break

When clearing bus break, do not specify the *Bus* and the following parameters.

Bus: Bus selection

`0` CPU system data bus

`1` CPU system instruction bus

`2` BBCU system bus

BBCU_BusMaster: BBCU bus master (decimal or hexadecimal)

Specify a value from 0x01 to 0x0f with the bit to be selected set to 1. (One or more conditions are selectable. At least one condition must be selected when *Bus* = 2.)

bit 3 = 1: Memory access by the USER (CPU)

bit 2 = 1: Memory access by the DMA

bit 1 = 1: Memory access for refill/write back by the Cache

bit 0 = 1: Memory access by the CPU

Trig/Break: Trigger/Break condition

`0` Trigger output only

`1` Trigger with break

R/W: Read/Write condition

`r` Read

`w` Write

`rw` Read/Write

Inst/Data: Instruction/Data condition

`0` Instruction

`1` Data

`2` Instruction/Data

Break by instruction (0 or 2) is effective only when the CPU is specified for *BBCU_BusMaster*.

Access: Access condition (decimal or hexadecimal)

Specify a value from 0x01 to 0x07 with the bit to be selected set to 1.

(One or more conditions are selectable.)

bit 2 = 1: Word access

bit 1 = 1: Half word access

bit 0 = 1: Byte access

Int: Interrupt condition

`0` Disables bus break by interrupt level

`1` Enables bus break by interrupt level

IntLevel: Interrupt level (decimal or hexadecimal)
0–15
Although the set value is ignored, any value must be set for IntLevel even if *Int* = 0 (disabled).

DataComp: Data comparison method
`0` Bus data is compared with 32-bit data.
`1` Bus data is compared with 16-bit data for each of 16 high-order and low-order bits then comparison results are ORed.

Address: Address (decimal or hexadecimal)
Specify within the range from 0x00000000 to 0xffffffff.

AddressMask: Address mask (decimal or hexadecimal)
Specify within the range from 0x00000000 to 0xffffffff.
Set the bits to be masked to 0. Address comparison between the bus and *Address* is performed only for the bits set to 1 in *AddressMask*.

Data: Data (decimal or hexadecimal)
Specify within the range from 0x00000000 to 0xffffffff.

DataMask: Data mask (decimal or hexadecimal)
Specify within the range from 0x00000000 to 0xffffffff.
Set the bits to be masked to 0. Data comparison between the bus and *Data* is performed only for the bits set to 1 in *DataMask*.

Counter: Break counter (decimal or hexadecimal)
Specify the break counter value within the range from 0x00000001 to 0x7fffffff. A break occurs when the break conditions are met the specified number of times.

CounterClr/Save: Break counter mode (decimal or hexadecimal)
`0` The break counter is not reset by program execution and break occurrence.
`1` The break counter is reset each time the program execution starts.

Usage example

■ Example 1

```
(gdb) c33 obb on 0 1 1 w 1 0x04 0 0 0 0 0 0x55555555 0xffffffff 3 0
Bus break on
Bus select                : CPU data bus
BBCU bus master select    : ---
Trigger/Break             : Break
Read/Write                : Write
Instruction/Data          : Data
Access size               : Word
Break by interrupt level : Disable
Interrupt level           : 0
Data compare              : 32bit
Address                   : 0
Address mask              : 0
Data                      : 0x55555555
Data mask                 : 0xffffffff
Break counter             : 3
Break counter Clear/Save : Save
```

The program being executed is made to break when a word data 0x55555555 is written through the CPU system data bus three times. The bus data is compared in 32-bit size.
When this break occurs, the following message is output:

```
Hardware on chip Bus Break!
```

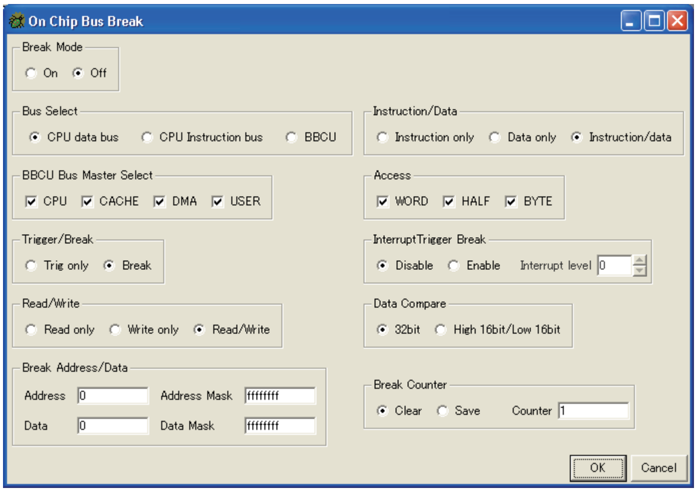
■ Example 2

```
(gdb) c33 obb off
Bus break off
```

The on-chip bus break conditions are cleared.

GUI

The bus break conditions can be set in the [On Chip Bus Break] dialog box. Select [Bus Break] from the [View] menu to display the dialog box.



On Chip Bus Break

Break Mode: ○ On ● Off

Bus Select: ● CPU data bus ○ CPU Instruction bus ○ BBCU

Instruction/Data: ○ Instruction only ○ Data only ● Instruction/data

BBCU Bus Master Select: ☑ CPU ☑ CACHE ☑ DMA ☑ USER

Access: ☑ WORD ☑ HALF ☑ BYTE

Trigger/Break: ○ Trig only ● Break

InterruptTrigger Break: ● Disable ○ Enable Interrupt level 0

Read/Write: ○ Read only ○ Write only ● Read/Write

Data Compare: ● 32bit ○ High 16bit/Low 16bit

Break Address/Data: Address 0 Address Mask ffffffff Data 0 Data Mask ffffffff

Break Counter: ● Clear ○ Save Counter 1

Break Mode:	Select [On] to set and enable on-chip bus break conditions or select [Off] to disable on-chip bus break.
Bus Select:	Select a bus.
BBCU Bus Master Select:	Select one or more BBCU bus masters. This selection is effective when [BBCU] is selected in the [Bus Select] field and is ineffective if another bus is selected.
Trigger/Break:	Select the operation when the break conditions are met, either to output a trigger without a break occurrence or to break program execution at the same time the trigger is output.
Read/Write:	Select the read/write condition.
Break Address/Data:	Input the address and data conditions and the mask bits used for the address/data comparison in hexadecimal values.
Instruction/Data:	Select the instruction/data condition. Break by instruction fetch is effective only when [CPU] is selected in the [BBCU Bus Master Select] field.
Access:	Select the access size condition.
Interrupt Trigger Break:	Select [Enable] and an interrupt level (0–15) from [Interrupt level] to include the interrupt level in the bus break conditions. Select [Disable] to remove the interrupt level from the bus break conditions.
Data Compare:	Select whether the data is compared in 32 bits or 16 high-order and low-order bits.
Break Counter:	Input the bus break count in the decimal number (1–2147483647). Furthermore, select whether the break counter is reset (Clear) each time the program execution starts or not (Save).
[OK] button:	When this button is clicked, the debugger sends the command with the values set here to ICD and closes the dialog box. If there is no selected check box in the [BBCU Bus Master Select] field, a message box with an error message is displayed.
[Cancel] button:	When this button is clicked, the debugger discards the above settings and closes the dialog box.

When the dialog box is reopened, it shows the previous set status. The first time the dialog box opens, it shows the default settings as the screen sample above.

Notes

- The `c33 obb` command is effective only in ICD3 and ICD6 modes.
- The `c33 obb` command has 16 parameters in the command line and they cannot be omitted. So we recommend use of the dialog box to specify parameters and enable a bus break.
  When executing the `c33 obb` from a command file, describe the command line in the file.
- When a break occurs by setting this command, the break counter value is displayed.
- Note that an error occurs if the target processor does not support the on-chip bus break function.
- The *Bus* parameter (or the [Bus Select] buttons on the dialog box) set in this command must be the same as that set in the bus trace command. If different bus conditions are specified between the commands, the bus condition set by the most recently executed command is effective.

c33 timebrk (set lapse of time break) [ICD6]

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

`c33 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) c33 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) c33 timebrk 0
      timer break off.
```

The lapse of time break is disabled.

Notes

- This command can be used only in ICD6 mode.
- When a break time has been set once, the lapse of time break is effective until it is disabled with "`c33 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.

c33 savebreak (save set breakpoints) [ICD2/ICD3/ICD6/SIM/MON]

Operation

Saves all set breakpoints in a file (CURRENT PATH\break.gdb).
Settings saved by this command are source file names and line numbers.
Breakpoints can be reset by the `c33 loadbreak` command after the .gdb file is closed and reloaded.

Format

```
c33 savebreak
```

Usage example

```
break main.c:50
break *0x80a2
break sub5
c33 savebreak
```

If addresses and symbol names are specified, they are replaced with source file names and line numbers before saving.

main.c:50 → source file name main.c line number: 50
*0x80a2 → source file name sub.c line number: 20
sub5 → source file name sub5.c line number: 15

< Saved breakpoints >

Software breakpoint
Temporary software breakpoint
Hardware breakpoint
Temporary hardware breakpoint
Breakpoints are saved only when they are valid.
If the number of breakpoints is 0, this command stores "breakpoints = 0".

Notes

- The following break settings are not saved.
  Data breakpoint
  On-chip area break
  On-chip bus break
- Every time this command is executed, the contents stored in the .gdb file are updated.
- This command generates an error if the target command has not been executed previously.
  Error message: "`C33 command error, command is not supported in present mode.`"
- The breakpoints set in C source files are not saved if the addresses of set breakpoints do not match the beginning addresses on the source level lines when displayed in SOURCE mode.

  Example:

  < Breakpoint saved >

  *0x80a2 → source file name sub.c line number 20 (0x80a2 matches the 20th line of sub.c)

  < Breakpoint not saved >

  *0x80a4 → source file name sub.c line number Address 20+2 (0x80a4 does not match the 20th line of sub.c)

c33 loadbreak (reload saved breakpoints) [ICD2/ICD3/ICD6/SIM/MON]

Operation

Resets breakpoints according to the contents of the file (CURRENT PATH\break.gdb) in which breakpoints were saved using the `c33 savebreak` command.

Format

```
c33 loadbreak
```

Usage example

```
target sim
load
c33 loadbreak
```

Notes

- This command generates an error if the target command has not been executed previously.
  Error message: "`C33 command error, command is not supported in present mode.`"
- In the following cases and at the following points, an error message does not appear and breakpoints are not reset.
  1. There is no applicable source file.
  2. There is no applicable line number.
  3. Lines of extended instructions, except the ext instruction at the beginning.
  4. Lines of delay instructions (lines next to delayed branch instructions).
  5. Breakpoints set at existing breakpoints.
  6. Software breaks exceed 200 locations.
  7. The number of valid hardware breaks exceeds the maximum number (1 to 4, depending on model).
  8. Software breaks set in ROM in the ICD mode.
  9. Breaks set at a location other than the parameter file in the SIM mode.
- When set in a C source line not expanded to a mnemonic, a breakpoint is not set in that line, but is instead set to the address of the first instruction to be executed thereafter.

  Example: When a breakpoint is set to main.c:2, it is set to main.c:5.

```
main.c
1:/*  Comment 1
2:    Comment 2
3:    Comment END */
4:    ucData = 1;
5:    for( i  = 0; i < 10; ++i )
6:    {
```

- If the `c33 savebreak` command has not been executed or the number of set breakpoints is 0 although the `c33 savebreak` command has been executed, nothing will happen when this command is executed.
- Also refer to the "Notes" for break, tbreak, hbreak, and thbreak commands.

10.7.9 Symbol Information Display Commands

info locals (display local symbol)
info var (display global symbol) [ICD2 / ICD3 / ICD6 / SIM / MON]

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) [ICD2 / ICD3 / ICD6 / SIM / MON]

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 ≤ *Value* ≤ 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 0x100000000 for variable int is processed as 0x00000000.

10.7.10 File Loading Commands

file (load debugging information) [ICD2 / ICD3 / ICD6 / SIM / MON]

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) c33 rpf sample.par (sets map information.)
(gdb) file sample.elf     (this command)
(gdb) target sim          (connects the target.)
(gdb) load                (loads the program.)
(gdb) c33 rsth            (hot reset)
```

- 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) [ICD2 / ICD3 / ICD6 / SIM / MON]

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) c33 rpf sample.par (sets map information.)
(gdb) file sample.elf    (loads debugging information.)
(gdb) target sim         (connects the target.)
(gdb) load               (this command)
(gdb) c33 rsth           (hot reset)
```

- 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.

10.7.11 Map Information Commands

c33 rpf (set map information) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Sets memory map information by loading a specified parameter file.

Format

`c33 rpf` ***Filename***

Filename: Name of parameter file (with path also specifiable)

Usage example

```
(gdb) c33 rpf sample.par
```

Memory map information is loaded from `sample.par` in the current directory.

Notes

- For details about parameter files, see Section 10.8, "Parameter Files".
- Make sure the `c33 rpf` command is executed only once prior to the `target` command. The following shows the basic sequence of command execution:

  ```
  (gdb) c33 rpf sample.par (this command)
  (gdb) file sample.elf    (loads debugging information.)
  (gdb) target sim         (connects the target.)
  (gdb) load               (loads the program.)
  (gdb) c33 rsth           (hot reset)
  ```

  To reset memory map information, quit the debugger temporary, then restart it and execute the `c33 rpf` command.
- The debugger reserves an area in computer memory according to memory map information loaded from a parameter file. If a necessary area cannot be reserved (with error message "Cannot allocate memory." displayed), correct area sizes in the parameter file less than 256MB per area or divide an area into two or more areas.
- No software PC breaks and temporary software PC breaks can be set in areas assigned the "ROM" attribute in a parameter file (`*.par`).

c33 map (display map information) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Displays memory map information set by a parameter file.

Format

```
c33 map
```

Usage example

```
(gdb) c33 map
CPU : Standard
Memory map information Type  Wait(r/w) Size  Endian
    00000000-000007ff  RAM     7/7     16Bit Little
    00040000-0004ffff  IO      7/7     16Bit Little
    00200000-002fffff  RAM     7/7     16Bit Little
    00600000-006fffff  RAM     7/7     16Bit Little
    00c00000-00cfffff  ROM     7/7     16Bit Little
    00600000-006fffff  STACK
```

CPU: Indicates the type of CPU.
`Standard`: C33 STD
`Advanced`: S1C33401 (C33 ADV)
`PE`: C33 PE

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 `c33 rpf` command, a message appears like the one shown below.

```
(gdb) c33 map
CPU : Standard
Memory map information Type  Wait(r/w) Size  Endian
No data.
```

10.7.12 Flash Memory Manipulation Commands

c33 fls (set flash memory) [ICD2 / ICD3 / ICD6 / MON]

Operation

Sets up flash memory of the target system as required to write data to it.

Format

`c33 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 ≤ *StartAddr* ≤ *EndAddr* ≤ 0xffffffff, 0 ≤ *ErasePrg* ≤ 0xffffffff, 0 ≤ *WritePrg* ≤ 0xffffffff

Usage example

```
(gdb) c33 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. For details about the flash write program, refer to the contents of the `utility\fls33\` directory.
- The specified address is only effective for the 32 low-order bits, with excessive bits being ignored. For example, address 0x100000000 is processed as 0x00000000.

c33 fle (erase flash memory) [ICD2 / ICD3 / ICD6 / MON]

Operation

Erases the contents of flash memory of the target system.

Format

`c33 fle` ***ControlReg StartBlock EndBlock*** `[`***Timer***`]`

ControlReg: Start address set by `c33 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) c33 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 `c33 fls` command. If you execute the `c33 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 Commands

c33 tm (set PC trace mode) [ICD2 / ICD3 / ICD6] [SIM]

Note: Note that the `c33 tm` command operates differently and uses different formats in ICD mode and simulator mode. Each mode is explained separately below.

ICD mode

Operation (ICD2/ICD3/ICD6)

In ICD mode, set trace mode, trigger addresses, and other trace conditions. The items to be set are listed below.

Trace mode

One of the following three trace modes can be set:

1. All trace mode, with overwrite

   A trace begins at program startup, with trace information written to trace memory of the S5U1C33000H or S5U1C33001H. When trace memory of the S5U1C33000H or S5U1C33001H is full, old data is overwritten beginning with the oldest. Therefore, trace memory always retains the latest trace information.

2. All trace mode, without overwrite

   A trace begins at program startup, with trace information written to trace memory of the S5U1C33000H or S5U1C33001H. Writing to trace memory of the S5U1C33000H or S5U1C33001H continues until the trace memory is full, with no more information written to it thereafter.

3. Area trace mode

   Only when a trace is executed in the range from trigger address 1 to trigger address 2, trace information is written to trace memory of the S5U1C33000H or S5U1C33001H.

Displaying the number of run cycles

You can choose to display the number of clock cycles (Clk) in trace information as the number of clock cycles executed for each instruction or as a cumulative count of cycles executed since a trace began.

Setting area trace mode conditions

The following conditions can be specified in area trace mode:

1. Trigger address

   Specify the start address (trigger address 1) and end address (trigger address 2) of the area to be traced.

2. Break at trigger address 2

   You can specify whether to break program execution at trigger address 2 or continue.

3. Time measurement condition

   You can choose to measure program execution time from start to break or measure execution time in only the trace area from trigger address 1 to trigger address 2.

Format (ICD2/ICD3/ICD6)

`c33 tm` ***Mode CycleInfo*** [***Trigger1 Trigger2 Break Measure***]

Mode: Trace mode
- `1` All trace mode, without overwrite
- `2` All trace mode, with overwrite (default)
- `3` Area trace mode

CycleInfo: Content of number of run cycles displayed
- `1` Cumulative cycles from start of a trace (default)
- `2` Individual cycles for each instruction

Trigger1: Start address of area trace (decimal, hexadecimal, or symbol)
Trigger2: End address of area trace (decimal, hexadecimal, or symbol)
Break: Break after area trace
- `1` Do not break at *Trigger2*.
- `2` Break at *Trigger2*.

Measure: Area trace time measurement
- `1` Measure execution time from start to break.
- `2` Measure execution time in trace area only.

Conditions: 0 ≤ *Trigger1* ≤ 0xffffffff, 0 ≤ *Trigger2* ≤ 0xffffffff

Always be sure to set *Trigger1*, *Trigger2*, *Break* and *Measure* for area trace mode. Do not specify these parameters for all trace mode.

Usage example (ICD2/ICD3/ICD6)

■ Example 1

```
(gdb) c33 tm 2 2
```

All trace mode with overwrite is specified, with the number of run cycles to be displayed for each instruction.

■ Example 2

```
(gdb) c33 tm 3 1 0x600000 0x600100 2 2
```

Area trace mode from address 0x600000 to address 0x600100 is specified, with the number of run cycles to be displayed as a cumulative count, program execution specified to break after trace, and execution time measured only in the trace area.

Trace information (ICD2/ICD3/ICD6)

When you run the target program continuously after setting trace mode with this command, trace information is sampled in trace memory of the S5U1C33000H or S5U1C33001H. After a break, you can display sampled trace information in the [Trace] window by using the `c33 td` command. Otherwise, you can write the information to a file by using the `c33 tf` command.
The contents of trace information displayed in a window are as follows:

```
Cycle  Address Code Unassemble        Clk  Method File          Line     SourceCode
000044 0C00004 C020 ext    0x20       000000 SPC (./boot.s)     00009    xld.w ...
000043 0C00006 6C0F ld.w   %r15,0x0   000016 SPC
000042 0C00008 A0F1 ld.w   %sp,%r15   000032 SPC (./boot.s)     00010    ld.w  ...
                                :
000003 0C00046 2E54 ld.w   %r4,%r5    000776 DPC
000002 0C00048 7014 and    %r4,0x1    000792 DPC
000001 0C0004A 6804 cmp    %r4,0x0    000808 DPC
000000 0C0004C 1809 jreq   0x9        000824 DPC
```

Cycle: Trace cycles (decimal)
The trace cycle for the last information sampled in trace memory is 000000.
Address: Addresses of instructions executed by the CPU (hexadecimal)
Code: Instruction codes executed by the CPU (hexadecimal)
Unassemble: Disassembled contents of instruction code
Clk: Number of clock cycles executed
This information can be displayed as cumulative clock cycles since the trace began or as clock cycles for each instruction executed, as specified by *CycleInfo*. (Cumulative clock cycles are shown above.)
Method: Method of trace analysis (or how to get instruction addresses)
`SPC`: Analysis by initial PC
`TRG`: Analysis by trigger address
`DPC`: Analysis by DPCO signal
`RET`: Analysis by `call` and `ret` statements
`MAP`: Analysis by map information
`RTI`: Analysis by `reti` statement
`---`: Analysis not possible
File: Name of source file containing executed instructions
Line: Source line numbers
SourceCode: Source codes

Notes (ICD2/ICD3/ICD6)

- When area trace mode is set, hardware PC breaks are disabled and ineffective. However, the hardware PC breakpoints set remain intact and are reenabled when the debugger exits area trace mode.
- The trigger addresses are only effective for the 32 low-order bits, with excessive bits being ignored. For example, address 0x100000000 is processed as 0x00000000. Moreover, if an address is specified with an odd value, the specified address is adjusted to the 16-bit boundary by assuming LSB = 0.

Simulator mode

Operation (SIM)

In simulator mode, only the parameters below can be set.

Turning trace on/off

When you turn trace on, trace information is sampled along with program execution.

Displaying register values and source codes

In addition to basic trace information, you can choose to display the contents of registers and source codes.

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 (SIM)

`c33 tm on` ***Mode*** [***Filename***]   (sets trace mode.)
`c33 tm off`   (clears trace mode.)

Mode: Trace mode (contents of trace information displayed)

- **`1`** Normal display
- **`2`** Register display
- **`3`** Normal display including source information
- **`4`** Register display including source information

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 (SIM)

■ Example 1

```
(gdb) c33 tm on 4 trace.log
```

Example for operation in simulator mode, where "register display including source information" is set as a trace option and file `trace.log` is specified as the destination at which to save 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) c33 tm off
```

Trace mode is turned off while operating in simulator mode. From this time on, no trace information is sampled even when running the program.

Trace information (SIM)

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:

Mode = 1 (normal display)

```
00000001 00C00004 C020  ext    0x20             -------- -- -------- 00000008
00000002 00C00006 6C0F  ld.w   %r15,0x0         -------- -- -------- 00000016
00000003 00C00008 A0F1  ld.w   %sp,%r15         -------- -- -------- 00000024
                                                         :
00000015 00C00026 6C04  ld.w   %r4,0x0          -------- -- -------- 00000150
00000016 00C00028 5C04  ld.w   [%sp+0x0],%r4    000007F8 wW 00000000 00000173
```

Mode = 2 (register display)

```
00000001 00C00004 C020  ext   0x20           -------- -- -------- 00000008
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA
 0AAAAAA8 00000000 00000000         IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000002 00C00006 6C0F  ld.w  %r15,0x0       -------- -- -------- 00000016
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA 00000800
 0AAAAAA8 00000000 00000000         IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000003 00C00008 A0F1  ld.w  %sp,%r15       -------- -- -------- 00000024
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA 00000800
 00000800 00000000 00000000         IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
                                          :
00000015 00C00026 6C04  ld.w  %r4,0x0        -------- -- -------- 00000150
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA 00000000 AAAAAAAA AAAAAAAA AAAAAAAA
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA 00000000
 000007F8 00000000 00000000         IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000016 00C00028 5C04  ld.w  [%sp+0x0],%r4  000007F8 wW 00000000 00000173
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA 00000000 AAAAAAAA AAAAAAAA AAAAAAAA
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA 00000000
 000007F8 00000000 00000000         IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
```

Mode = 3 (normal display including source information)

```
00000001 00C00004 C020  ext   0x20           -------- -- -------- 00000008 (./boot.s)  00009  xld.w ...
00000002 00C00006 6C0F  ld.w  %r15,0x0       -------- -- -------- 00000016
00000003 00C00008 A0F1  ld.w  %sp,%r15       -------- -- -------- 00000024 (./boot.s)  00010  ld.w  ...
                                          :
00000015 00C00026 6C04  ld.w  %r4,0x0        -------- -- -------- 00000150 (./main.c)  00011  for   ...
00000016 00C00028 5C04  ld.w  [%sp+0x0],%r4  000007F8 wW 00000000 00000173
```

Mode = 4 (register display including source information)

```
00000001 00C00004 C020  ext   0x20           -------- -- -------- 00000008 (./boot.s)  00009  xld.w ...
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA
 0AAAAAA8 00000000 00000000         IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000002 00C00006 6C0F  ld.w  %r15,0x0       -------- -- -------- 00000016
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA 00000800
 0AAAAAA8 00000000 00000000         IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000003 00C00008 A0F1  ld.w  %sp,%r15       -------- -- -------- 00000024 (./boot.s)  00010  ld.w  ...
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA 00000800
 00000800 00000000 00000000         IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
                                          :
00000015 00C00026 6C04  ld.w  %r4,0x0        -------- -- -------- 00000150 (./main.c)  00011  for   ...
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA 00000000 AAAAAAAA AAAAAAAA AAAAAAAA
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA 00000000
 000007F8 00000000 00000000         IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000016 00C00028 5C04  ld.w  [%sp+0x0],%r4  000007F8 wW 00000000 00000173
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA 00000000 AAAAAAAA AAAAAAAA AAAAAAAA
 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA 00000000
 000007F8 00000000 00000000         IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
```

<First line of each item of trace information>

Number Address Code Disassemble Address Type Data Clock [*File Line SourceCode*]

Number:	Number of executed instructions (decimal) Number of instructions executed since the CPU was reset or trace turned on
Address:	Address of executed instructions (hexadecimal)
Code:	Instruction codes (hexadecimal)
Disassemble:	Disassembled contents of executed instructions
Address:	Accessed memory addresses (hexadecimal)
Type:	Type of bus operation `rB`: Byte data read; `rH`: Half word data read; `rW`: Word data read `wB`: Byte data write; `wH`: Half word data write; `wW`: Word data write
Data:	Read/write data (hexadecimal)
Clock:	Number of clock cycles executed (decimal)
File:	Source file names (only when source display is selected by `c33 tm` command)
Line:	Source line numbers (only when source display is selected by `c33 tm` command)
SourceCode:	Source codes (only when source display is selected by `c33 tm` command)

<2nd to 4th (6th) lines of each item of trace information>

These lines are only displayed when you choose to display the contents of registers. Register values are displayed in order of the following:

For the C33 STD Core

```
R0   R1   R2   R3   R4   R5   R6   R7
R8   R9   R10  R11  R12  R13  R14  R15
SP   AHR  ALR       PSR  (displayed individually for each flag.)
```

For the C33 ADV Core

```
R0   R1   R2   R3   R4   R5   R6   R7
R8   R9   R10  R11  R12  R13  R14  R15
SP   AHR  ALR       PSR  (displayed individually for each flag.)
LCO  LSA  LEA  SOR  TTBR DP   USP  SSP
PSR  (displayed individually for each flag added to the C33 ADV Core.)
```

Notes (SIM)

- 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 `c33 tm off`), then set a new trace mode.

c33 td (display PC trace content) [ICD2 / ICD3 / ICD6]

Operation

Displays trace information saved in trace memory of the S5U1C33000H or S5U1C33001H in the [Trace] window. Note that executing this command opens the [Trace] window if closed.

Format

`c33 td` [*StartCycle EndCycle*]

StartCycle: Trace cycle number from which to start display (decimal or hexadecimal)
EndCycle: Trace cycle number at which to complete display (decimal or hexadecimal)
Cycle number 0 is the latest data. If *StartCycle* and *EndCycle* are omitted, trace data for 10K clock cycles are displayed.
Conditions: S5U1C33000H $0 \le EndCycle \le StartCycle \le 131{,}071$
S5U1C33001H $0 \le EndCycle \le StartCycle \le 1{,}048{,}574$

Usage example

■ Example 1

```
(gdb) c33 td
```

The trace information for 10K clock cycles in the trace memory is displayed in the [Trace] window.

■ Example 2

```
(gdb) c33 td 1000 0
```

The trace information from cycle number 1,000 to the latest cycle is displayed in the [Trace] window.

Notes

- This command can only be used in ICD2, ICD3, and ICD6 modes.
- If search mode is set by the `c33 ts` command, only trace data that matches search conditions in a specified range of trace cycles is displayed.
- Before this command can be used to get trace information, memory map information must be set by the `c33 rpf` command. Note, however, that the `c33 rpf` command must be executed before the `target` command. If memory map information is set immediately before the program starts running, trace information cannot be acquired normally.
  The following shows an example sequence of command execution:

```
(gdb) c33 rpf sample.par (sets map information.)
(gdb) file sample.elf    (loads debugging information.)
(gdb) target icd com1    (connects the target.)
(gdb) load               (loads the program.)
(gdb) c33 rsth           (hot reset)
(gdb) c33 tm 1 1         (sets ICD trace mode.)
(gdb) break 13           (sets breakpoint.)
(gdb) continue           (runs the program continuously.) ... trace information acquired
(gdb) c33 td             (displays trace information.)
```

c33 ts (search PC trace content) [ICD2 / ICD3 / ICD6]

Operation

Sets search mode to display and save trace information for only the cycle in which the instruction at a specified address was executed.

Format

`c33 ts` ***Address*** [***PreLine PostLine***]

Address: Executed address (decimal, hexadecimal, or symbol)
Note that specifying 0xffffffff turns off search mode.
PreLine: Number of lines to display and save before searched line (decimal or hexadecimal)
PostLine: Number of lines to display and save after searched line (decimal or hexadecimal)
Conditions: $0 \le Address \le 0xffffffff$, $0 \le PreLine \le 256$, $0 \le PostLine \le 256$

Usage example

■ Example 1

```
(gdb) c33 ts 0x20100 5 5
```

The trace information for only the cycle in which the instruction at address 0x20100 was executed, and for the five cycles before and after are displayed and saved.

■ Example 2

```
(gdb) c33 ts 0xffffffff
```

Search mode is turned off.

Notes

- This command can only be used in ICD2, ICD3, and ICD6 modes.
- Before this command can be used to get trace information, memory map information must be set by the `c33 rpf` command. Note, however, that the `c33 rpf` command must be executed before the `target` command. If memory map information is set immediately before the program starts running, trace information cannot be acquired normally.
  The following shows an example sequence of command execution:

```
(gdb) c33 rpf sample.par (sets map information.)
(gdb) file sample.elf    (loads debugging information.)
(gdb) target icd com1    (connects the target.)
(gdb) load               (loads the program.)
(gdb) c33 rsth           (hot reset)
(gdb) c33 tm 1 1         (sets ICD trace mode.)
(gdb) break 13           (sets breakpoint.)
(gdb) continue           (runs the program continuously.) ... trace information acquired
(gdb) c33 ts 0x20100 5 5 (searches for trace information.)
```

c33 tf (save PC trace content) [ICD2 / ICD3 / ICD6]

Operation

Saves the trace information in trace memory of the S5U1C33000H or S5U1C33001H to a file. The same contents are saved to a file as those displayed in a window by the `c33 td` command.

Format

`c33 tf` ***Filename*** [***StartCycle EndCycle***]

Filename: Name of file in which to save trace information
StartCycle: Trace cycle number from which to start saving (decimal or hexadecimal)
EndCycle: Trace cycle number at which to complete saving (decimal or hexadecimal)
Cycle number 0 is the latest data. If *StartCycle* and *EndCycle* are omitted, trace data for 10K clock cycles are saved.
Conditions: S5U1C33000H $0 \leq EndCycle \leq StartCycle \leq 131,071$
S5U1C33001H $0 \leq EndCycle \leq StartCycle \leq 1,048,574$

Usage example

■ Example 1

```
(gdb) c33 tf trace.log
```

The trace information for 10K clock cycles in the trace memory is saved to the `trace.log` file.

■ Example 2

```
(gdb) c33 tf trace.log 1000 0
```

The trace information from cycle number 1,000 to the latest cycle is saved to the `trace.log` file.

Notes

- This command can only be used in ICD2, ICD3, and ICD6 modes.
- If search mode is set by the `c33 ts` command, only the trace data that matches search conditions in a specified range of trace cycles is written to a file.
- If an existing file is specified, it is overwritten with data.
- Before this command can be used to get trace information, memory map information must be set by the `c33 rpf` command. Note, however, that the `c33 rpf` command must be executed before the `target` command. If memory map information is set immediately before the program starts running, trace information cannot be acquired normally.
  The following shows an example sequence of command execution:

```
(gdb) c33 rpf sample.par (sets map information.)
(gdb) file sample.elf    (loads debugging information.)
(gdb) target icd com1    (connects the target.)
(gdb) load               (loads the program.)
(gdb) c33 rsth           (hot reset)
(gdb) c33 tm 1 1         (sets ICD trace mode.)
(gdb) break 13           (sets breakpoint.)
(gdb) continue           (runs the program continuously.) ... trace information acquired
(gdb) c33 tf trace.log   (saves trace information.)
```

- Because trace memory of the S5U1C33001H is large, it takes time to save all trace data to a file. We recommend that you specify a cycle number or set search mode before saving.

c33 obt (set on-chip bus trace) [ICD3 / ICD6]

Operation

Configures the bus trace conditions. The bus trace function acquires the bus cycle information and saves it in the trace memory when an access in the specified condition occurs on the specified bus.

This command is effective only when the target processor incorporates the C33 ADV Core.

The bus trace function is disabled at **gdb** startup. To enable bus trace, execute this command by setting the *Mode* parameter to 0 or 1.

Format

`c33 obt` ***Mode Bus BBCU_BusMaster R/W Inst/Data Access BusOutput AddressBusSize Compress UserInput***

Mode: Bus trace mode enable
- `0` Enables normal trace mode
- `1` Enables user trace mode
- `2` Disables bus trace function (default)

Bus: Bus selection
- `0` CPU system data bus (default)
- `1` CPU system instruction bus
- `2` BBCU system bus

Set 0 in user mode as this parameter is ineffective.

BBCU_BusMaster: BBCU bus master (decimal or hexadecimal)

Specify a value from 0x01 to 0x0f with the bit to be selected set to 1. (One or more conditions are selectable. At least one condition must be selected when Bus = 2.)

bit 3 = 1: Memory access by the USER (CPU) (default)
bit 2 = 1: Memory access by the DMA (default)
bit 1 = 1: Memory access at the time of the refill/light back by the Cache (default)
bit 0 = 1: Memory access of the CPU (default)

Set 0 in user mode as this parameter is ineffective.

R/W: Read/Write condition
- `r` Read
- `w` Write
- `rw` Read/Write (default)

Inst/Data: Instruction/Data condition
- `0` Instruction
- `1` Data
- `2` Instruction/Data (default)

Access: Access conditions (decimal or hexadecimal)

Specify a value from 0x01 to 0x07 with the bit to be selected set to 1. (One or more conditions are selectable.)

bit 2 = 1: Word access (default)
bit 1 = 1: Half word access (default)
bit 0 = 1: Byte access (default)

BusOutput: Bus output condition
- `0` Address bus
- `1` Data bus
- `2` Address bus/Data bus (default)

AddressBusSize: Address bus output size

Specify the address information size in bytes (number of bytes from the lowest byte) that will be output when Address bus (0) is selected for the BusOutput parameter. The specifiable range is one to four (four bytes in default). Although the set data is ineffective, set any value even if the address bus output is disabled.

Compress: Compression output conditions
- `0` Not compressed.
- `1` Addresses and data are compressed when they are output. (default)

UserInput: User input condition (break trigger)
- `0` External trace signal input (TRC IN) is used to break. (default)
- `1` Target CPU trigger signal input (DBT: debug break trigger signal) is used to break.

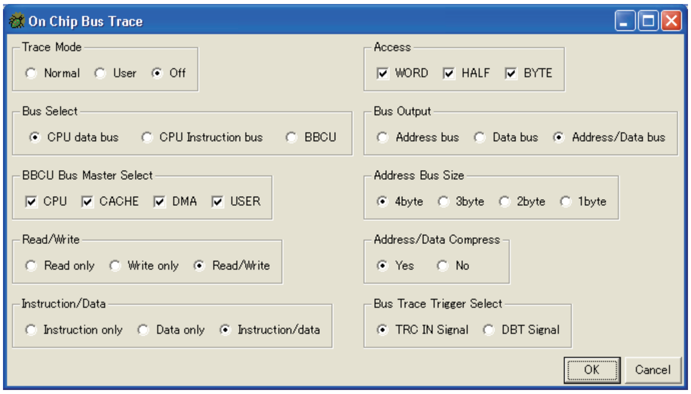
Usage example

```
(gdb) c33 obt 0 2 0x8 rw 2 0x04 0 4 0 0
Set bus trace
Trace mode                : Normal mode
Bus select                : BBCU
BBCU bus master select    : USER
Read/Write                : Read/Write
Instruction/Data          : Instruction/Data
Access size               : Word
Bus output                : Address bus
Address bus size          : 4 byte
Compression               : Disable
Bus trace trigger select  : TRC IN signal
```

The bus trace conditions are set as follows:
Word access by USER, 4-byte address output without compression, and using an external trace input (TRC IN) signal as the user input break trigger signal.

GUI

The bus trace conditions can be set in the [On Chip Bus Trace] dialog box. Select [Bus Trace] from the [View] menu to display the dialog box.


On Chip Bus Trace

Trace Mode: ○ Normal ○ User ● Off

Access: ☑ WORD ☑ HALF ☑ BYTE

Bus Select: ● CPU data bus ○ CPU Instruction bus ○ BBCU

Bus Output: ○ Address bus ○ Data bus ● Address/Data bus

BBCU Bus Master Select: ☑ CPU ☑ CACHE ☑ DMA ☑ USER

Address Bus Size: ● 4byte ○ 3byte ○ 2byte ○ 1byte

Read/Write: ○ Read only ○ Write only ● Read/Write

Address/Data Compress: ● Yes ○ No

Instruction/Data: ○ Instruction only ○ Data only ● Instruction/data

Bus Trace Trigger Select: ● TRC IN Signal ○ DBT Signal

Trace Mode:	Select either [Normal] or [User] as the trace mode to set and enable on-chip bus trace conditions or select [Off] to disable on-chip bus trace.
Bus Select:	Select a bus.
BBCU Bus Master Select:	Select one or more BBCU bus masters. This selection is effective when [BBCU] is selected in the [Bus Select] field and is ineffective if another bus is selected.
Read/Write:	Select the read/write condition.
Instruction/Data:	Select the instruction/data condition. Tracing for instruction fetch is effective only when [CPU] is selected in the [BBCU Bus Master Select] field.
Access:	Select the access size condition.
Bus Output:	Select the bus output condition.
Address Bus Size:	Select the address bus output size. This setting is effective when [Address bus] or [Address/Data bus] is selected in the [Bus Output] field.
Address/Data Compress:	Select whether the bus trace data is compressed or not when output from the ICD.
Bus Trace Trigger Select:	Select a user input bus trace trigger signal from TRC IN and DBT.
[OK] button:	When this button is clicked, the debugger sends the command with the values set here to ICD and close the dialog box.
[Cancel] button:	When this button is clicked, the debugger discards the above settings and close the dialog box.

When the dialog box is reopen, it shows the previous set status. At the first time the dialog box opens, it shows the default settings as the screen sample above.

Notes

- The `c33 obt` command is effective only in ICD3 and ICD6 modes.
- The `c33 obt` command has 10 parameters in the command line and they cannot be omitted. So we recommend use of the dialog box to specify parameters and enable bus trace. When executing the `c33 obt` from a command file, describe the command line in the file.
- Note that an error occurs if the ICD trace function is disabled or the target processor does not support the on-chip bus trace function.
- The Bus parameter (or the [Bus Select] buttons on the dialog box) set in this command must be the same as that set in the bus break command. If different bus conditions are specified between the commands, the bus condition set by the recently executed command is effective.

c33 otd (display on-chip bus trace content) [ICD3 / ICD6]

Operation

Displays on-chip bus trace data in the [Trace] window.

Format

`c33 otd` ***Mode*** [***StartCycle EndCycle***]

Mode: Output mode
- `0` Outputs only bus trace information
- `1` Outputs PC trace and bus trace information

StartCycle: Trace cycle number from which to start display (decimal or hexadecimal)

EndCycle: Trace cycle number at which to complete display (decimal or hexadecimal)

Cycle number 0 is the latest data. If *StartCycle* and *EndCycle* are omitted, trace data for 10K clock cycles are displayed.

Conditions: 0 ≤ *EndCycle* ≤ *StartCycle* ≤ 1,048,574

Usage example

■ Example 1

```
(gdb) c33 otd 0
```

The bus trace information for 10K clock cycles in the trace memory is displayed in the [Trace] window.

■ Example 2

```
(gdb) c33 otd 1 1000 0
```

The PC trace and bus trace information from cycle number 1,000 to the latest cycle is displayed in the [Trace] window.

Display

Mode = 0

- Normal trace mode

```
Cycle   Clk     BM   Address  Data      I/D RW SZ UIN Unassemble
0000102 0000006 CPU           00000000   i   r  W   0  nop
                                                      nop
0000101 0000020 CPU  006002240           i   r  W   0
0000100 0000026 CPU  006000244           i   r  W   0
0000099 0000034 CPU  006000248           i   r  W   0
   :       :     :       :        :      :   :  :   :    :
```

The cycles without a trace data acquired are not displayed.

- User trace mode

```
Cycle   Clk     DTD       DTS    UIN
0009999 0000000 00000000  00000   0
0009998 0000001 00000000  00000   0
0009997 0000002 00000000  00000   0
0009996 0000003 00000000  00000   0
   :       :       :        :     :
```

Mode = 1

- Normal trace mode

```
Cycle   Address  Code Unassemble               Clk     BM   Address   Data      I/D RW SZ UIN ...
0000012 0060039E FC04 ld.w     [%dp+0x0],%r4   0000000                                        ...
                                               0000005 CPU  006003a8             i   r  W   1  ...
0000011 006003A0 2E40 ld.w     %r0,%r4         0000007                                        ...
                                               0000013 CPU  005003ac             i   r  W   1  ...
                                               0000021 CPU  005003b0             i   r  W   1  ...
   :       :      :     :           :             :     :      :        :        :   :  :   :  ...
```

- User trace mode

```
Cycle   Address  Code Unassemble               Clk     DTD       DTS     UIN Method File ...
0000012 0060039E FC04 ld.w     [%dp+0x0],%r4   0000000 10000000  00000    0    SPC      ...
                                               0000001 10000000  00000    0             ...
                                               0000002 10100000  00000    1             ...
                                               0000003 11100000  00001    1             ...
   :       :      :     :           :             :       :        :      :     :    :  ...
```

BM: Bus master
- `CPU` CPU (HBCU)
- `CPU/CACHE` CPU or Cache (BBCU)
- `DMA/USR` DMA or USER (BBCU)

Address: Address accessed

Data: Data accessed

I/D: Data type
- `i` Instruction
- `d` Data

RW: Read/Write
- `w` Write
- `r` Read

SZ: Access data size
- `B` Byte
- `H` Half word
- `W` Word

UIN: USER_IN signal status
- `0` or `1`

<Address/Data symbols>

*: Unknown value (address and data)

+: Unknown but the same value as the previous cycle (address only)

<Bit arrangement on the instruction bus>

Two instructions are fetched simultaneously through the instruction bus.

`ABCDEFGH`

`A` = I2[15:12]
`B` = I2[11: 8]
`C` = I2[7: 4]
`D` = I2[3: 0]
`E` = I1[15:12]
`F` = I1[11: 8]
`G` = I1[7: 4]
`H` = I1[3: 0]

Notes

- The `c33 otd` command is effective only in ICD3 and ICD6 modes.
- Before the `c33 otd` command can be used for debugging a C33 STD or C33 PE Core model, the `c33 logiana` command must be executed. In C33 STD and C33 PE Core models, the bus trace output is performed only in user mode.

c33 otf (save on-chip bus trace content) [ICD3 / ICD6]

Operation

Saves on-chip bus trace information to a file. The same contents are saved to a file as those displayed in the window by the `c33 otd` command.

Format

`c33 otf` ***Mode Filename*** [***StartCycle EndCycle***]

Mode:	Output mode **`0`** Outputs only bus trace information **`1`** Output PC trace and bus trace information
Filename:	Name of file in which to save trace information
StartCycle:	Trace cycle number from which to start saving (decimal or hexadecimal)
EndCycle:	Trace cycle number at which to complete saving (decimal or hexadecimal) Cycle number 0 is the latest data. If *StartCycle* and *EndCycle* are omitted, trace data for 10K clock cycles are saved.
Conditions:	$0 \leq EndCycle \leq StartCycle \leq 1{,}048{,}574$

Usage example

■ Example 1

```
(gdb) c33 otf 0 trace.log
```

The bus trace information for 10K clock cycles in the trace memory is saved to the `trace.log` file.

■ Example 2

```
(gdb) c33 otf 1 trace.log 1000 0
```

The PC trace and bus trace information from cycle number 1,000 to the latest cycle is saved to the `trace.log` file.

Notes

- The `c33 otf` command is effective only in ICD3 and ICD6 mode.
- Before the `c33 otf` command can be used for debugging a C33 STD or C33 PE Core model, the `c33 logiana` command must be executed. In C33 STD and C33 PE Core models, bus trace output is performed only in user mode.

10.7.14 Simulated I/O Commands

c33 stdin (data input simulation) [ICD2 / ICD3 / ICD6 / SIM / MON]

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 `c33 stdin` command:

• Break address	(position at which **gdb** takes in data)
• Input buffer address	(65-btye buffer)
• Input source	(file or [Simulated I/O] window)

For operation on the program side, see Section 10.6.8, "Simulated I/O".

Format

`c33 stdin 1` ***BreakAddr BufferAddr*** [***Filename***] (set)

`c33 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: $0 \le BreakAddr \le$ 0xffffffff, $0 \le BufferAddr \le$ 0xffffffff

Input examples

■ Example 1

```
(gdb) c33 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) c33 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) c33 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 `c33 stdin` command cannot duplicate those of software PC breaks. Be sure to clear software PC breaks before executing the `c33 stdin` command. Break addresses overlapping those of hardware PC breakpoints are accepted.
- The break and buffer addresses are only effective for the 32 low-order bits, with excessive bits being ignored. For example, address 0x100000000 is processed as 0x00000000. Moreover, if the break address is specified with an odd value, the specified address is adjusted to the 16-bit boundary by assuming LSB = 0.

c33 stdout (data output simulation) [ICD2 / ICD3 / ICD6 / SIM / MON]

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 `c33 stdout` command:

• Break address	(position at which **gdb** outputs data)
• Output buffer address	(65-btye buffer)
• Output destination	(file or [Simulated I/O] window)

For operation on the program side, see Section 10.6.8, "Simulated I/O".

Format

`c33 stdout 1` ***BreakAddr BufferAddr*** [***Filename***] (set)
`c33 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 ≤ *BreakAddr* ≤ 0xffffffff, 0 ≤ *BufferAddr* ≤ 0xffffffff

Input examples

■ Example 1

```
(gdb) c33 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) c33 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) c33 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 `c33 stdout` command cannot duplicate those of software PC breaks. Be sure to clear software PC breaks before executing the `c33 stdout` command. Break addresses overlapping those of hardware PC breakpoints are accepted.
- The break and buffer addresses are only effective for the 32 low-order bits, with excessive bits being ignored. For example, address 0x100000000 is processed as 0x00000000. Moreover, if the break address is specified with an odd value, the specified address is adjusted to the 16-bit boundary by assuming LSB = 0.
- Executing the `printf()` function may slow display in the [Simulated I/O] window. To increase display speed, you can add the C/C++ compiler option `-fno-builtin`.

10.7.15 Flash Writer Commands

c33 fwe (erase program/data) [ICD3 / ICD6]

Operation

This is a dedicated command for the flash writer of the S5U1C33001H.
It erases the data erase/write program or write data and address information loaded in the S5U1C33001H.

Format

```
c33 fwe 0      (erases write data.)
c33 fwe 1      (erases data/erase write program.)
```

Usage example

■ Example 1

```
(gdb) c33 fwe 0
```

The storage area for flash write data and the address information in the S5U1C33001H are erased.

■ Example 2

```
(gdb) c33 fwe 1
```

The storage area for the flash erase/write program and the entry information in the S5U1C33001H are erased.

Notes

This command can only be used in ICD3 and ICD6 modes.

c33 fwlp (load program) [ICD3 / ICD6]

Operation

This is a dedicated command for the flash writer of the S5U1C33001H.
It loads the data erase/write program from the host into the S5U1C33001H.

Format

`c33 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 ≤ *EraseEntryAddr* ≤ 0xfffffffe (A0 = 0), 0 ≤ *WriteEntryAddr* ≤ 0xfffffffe (A0 = 0), 0 ≤ comment size ≤ 128 bytes

Usage example

```
(gdb) c33 fwlp writer.sa 0x90 0xb4
```

The data erase/write program for the flash writer (prepared in file `writer.sa`) is loaded in the S5U1C33001H, and start addresses of the erase and write routines are set to 0x90 and 0xb4, respectively.

Notes

- This command can only be used in ICD3 and ICD6 modes.
- The data erase/write program should be 8 KB or less for ICD Ver. 3/Ver. 4 (ICD3 mode) or 64 KB or less for ICD Ver. 6 (ICD6 mode)

c33 fwld (load data) [ICD3 / ICD6]

Operation

This is a dedicated command for the flash writer of the S5U1C33001H.
It loads the data to be written to flash memory from the host into the S5U1C33001H.

Format

`c33 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 ≤ *EraseStartBlock* ≤ 0xffffffff, 0 ≤ *EraseEndBlock* ≤ 0xffffffff
If *EraseStartBlock* = *EraseEndBlock* = 0, all blocks in flash memory are assumed to be erased.
0 ≤ comment size ≤ 128 bytes

Usage example

```
(gdb) c33 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 S5U1C33001H.

Notes

- This command can only be used in ICD3 and ICD6 modes.
- The program is 8MB or less in size for ICD Ver. 3/Ver. 4 (ICD3 mode) or 4MB or less for ICD Ver. 6 (ICD6 mode).

c33 fwdc (copy target memory) [ICD3 / ICD6]

Operation

This is a dedicated command for the flash writer of the S5U1C33001H.
It loads the data stored in target board memory into the S5U1C33001H so that the data can be written to flash memory.

Format

`c33 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 ≤ *SourceAddr* ≤ 0xffffffe (A0 = 0), 0 ≤ *Size* ≤ 0xfffffffe (D0 = 0),
0 ≤ *EraseStartBlock* ≤ 0xffffffff, 0 ≤ *EraseEndBlock* ≤ 0xffffffff
When *EraseStartBlock* = *EraseEndBlock* = 0, all blocks in flash memory are assumed to be erased.
0 ≤ comment size ≤ 128 bytes

Usage example

```
(gdb) c33 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 S5U1C33001H.

Notes

- This command can only be used in ICD3 and ICD6 modes.
- The program is 8MB or less in size for ICD Ver. 3/Ver. 4 (ICD3 mode) or 4MB or less for ICD Ver. 6 (ICD6 mode).

c33 fwd (display flash writer information) [ICD3 / ICD6]

Operation

This is a dedicated command for the flash writer of the S5U1C33001H.
It displays information on the data and erase/write program loaded in the S5U1C33001H.

Format

`c33 fwd`

Display

```
(gdb) c33 fwd
CPU data address             : xxxxxxxx
Data size                    : xxxxxxxx
Erase start block            : xxxxxxxx
Erase end block              : xxxxxxxx
Erase parameter              : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

CPU program address          : xxxxxxxx
Program size                 : xxxxxxxx
Erase routine entry address : xxxxxxxx
Write routine entry address : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

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 `c33 fwld` or `c33 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 `c33 fwlp` command.

Notes

This command can only be used in ICD3 and ICD6 modes.

10.7.16 Other Commands

c33 log (logging) [ICD2 / ICD3 / ICD6 / SIM / MON]

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

`c33 log` ***Filename***	(starts logging.)
`c33 log`	(completes logging.)

Filename: Name of log file

Usage example

■ Example 1

```
(gdb) c33 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 `c33 log` command is subsequently executed.

■ Example 2

```
(gdb) c33 log
log off
```

The log file is closed and log output terminated.

Notes

- If an existing file name is specified, the `c33 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 `c33 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 `c33 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 `c33 log` command, because no log files will be created.

source (execute command file) [ICD2 / ICD3 / ICD6 / SIM / MON]

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 = `gwb33.cmd`>

```
# gwb33.cmd command file made by GWB33
# load symbol information
file /cygdrive/c/EPSON/gnu33/sample/tst/sample.elf
#decide connect mode and its port
target sim
# load to memory
load /cygdrive/c/EPSON/gnu33/sample/tst/sample.elf
# set ttbr register
set {char}0x4812d=0x59
set {long}0x48134=0x00C00000
# hot reset
c33 rsth
```

From # to the end of the line is interpreted as a comment.

```
(gdb) source gwb33.cmd
(gdb)
(gdb)
(gdb) file /cygdrive/c/EPSON/gnu33/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/gnu33/sample/tst/sample.elf
Loading section .text, size 0xbc lma 0xc00000
Start address 0xc00000
Transfer rate: 1504 bits in <1 sec.
(gdb)
(gdb) set {char}0x4812d=0x59
(gdb) set {long}0x48134=0x00C00000
(gdb)
(gdb) c33 rsth
CPU hot 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.

c33 clockmd (set execution counter mode)
c33 clock (display execution counter) [ICD2 / ICD3 / ICD6 / SIM]

Operation

`c33 clockmd`: Sets mode of the execution counter. The contents to be set consist of the following two items:

Counter mode

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).

Measurement unit

In ICD2, ICD3, and ICD6 modes, the unit of measurement used by the S5U1C33000H or S5U1C33001H execution counter can be selected from cycles, µs, or seconds. In simulator mode, the execution counter counts only the number of cycles. In debug monitor mode, this measurement facility is not available.

`c33 clock`: Displays the results counted during program execution.

For details about the execution counter, see "Measuring the execution cycles/execution time" in Section 10.6.5, "Executing the Program".

Format

`c33 clockmd` ***Mode Function*** (sets execution counter mode.)
`c33 clock` (displays execution counter.)

Mode: Counter mode
`1` Reset mode
`2` Cumulating mode (default)
Function: Unit of measurement
`1` Number of cycles (default) * In simulator mode, only option 1 can be selected.
`2` Units of µs
`3` Units of seconds

Usage example

■ Example 1

```
(gdb) c33 clockmd 2 1
(gdb) c33 rsth
CPU hot resetting ........ done
(gdb) continue
Continuing.

Breakpoint 1, sub (k=0) at ./main.c:20
(gdb) c33 clock
           218 cycle
(gdb) continue
Continuing.

Breakpoint 1, sub (k=1) at ./main.c:20
(gdb) c33 clock
           330 cycle
```

After being set to measure the number of cycles in cumulating mode, the execution counter is reset by hot-resetting the CPU, thereby starting the program. In cumulating mode, the execution counter is not reset even when program execution is resumed after a break.

■ Example 2

```
(gdb) c33 clockmd 1 1
(gdb) c33 rsth
CPU hot resetting ........ done
(gdb) continue

Breakpoint 1, sub (k=0) at ./main.c:20
(gdb) c33 clock
           218 cycle
(gdb) continue
Continuing.

Breakpoint 1, sub (k=1) at ./main.c:20
(gdb) c33 clock
           112 cycle
```

In this example, after being set to measure the number of cycles in reset mode, the execution counter is reset by hot-resetting the CPU, 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.

The count values displayed by the `c33 clock` command differ depending on the unit of measurement specified, as shown below.

```
(gdb) c33 clock
9999999999 cycle       (number of cycles)

(gdb) c33 clock
999999999.99 us        (units of µs)

(gdb) c33 clock
9999.999999 s          (units of seconds)
```

Notes

- In debug monitor mode, this command cannot be used because the execution counter is not supported.
- If the program is run for a long time exceeding the maximum value of the counter, a message ("clock timer overflow") appears.
- The execution counter is reset in the following cases:
    1. When the `c33 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
       In reset mode, if the program is single-stepped after setting the number of steps to be executed = 1 (or single-stepped using a menu command or toolbar button), the counter is reset each time one step of the program is executed.
    3. When the CPU is cold-reset or hot-reset

target (connect target) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Establishes connection to the target and sets connect mode.

ICD2 mode:	Connected with the S5U1C33000H (ICD ver. 2) via a serial interface.
ICD3 mode:	Connected with the S5U1C33001H1100 (ICD ver. 3) or S5U1C33001H1200 (ICD ver. 4) via a USB interface.
ICD6 mode:	Connected with the S5U1C33001H1400 (ICD ver. 6) via a USB interface.
Simulator mode:	Debugger is set to simulator mode.
Debug monitor mode:	Connected with the S5U1C330M1D1 (+ S5U1C330M2S) via a serial interface.

Format

`target` ***Type*** [***Time***]

Type: One of the following symbols that specify the target

`icd usb`: Connected with the S5U1C33001H1100 (ICD ver. 3) or S5U1C33001H1200 (ICD ver. 4) via a USB interface (in ICD3 mode).

`icd usb2`: Connected with the S5U1C33001H1100 (ICD ver. 3) or S5U1C33001H1200 (ICD ver. 4) via a USB interface (in ICD3 mode).
This is used when debugging the target in an environment where a single PC and two ICD ver. 3/ver. 4 units are connected via USB.
The debugger must be started up with the `--double_starting` specified and use "`target icd usb2`" to connect the debugger.
This allows you to start two debuggers with a single PC.

`icd6 usb`: Connected with the S5U1C33001H1400 (ICD ver. 6) via a USB interface (in ICD6 mode).

`icd com`*N*: Connected with the S5U1C33000H (ICD ver. 2) or S5U1C330M1D1 (+ S5U1C330M2S) via a serial interface (in ICD2 or MON mode), where *N* denotes a port number that can be selected from com1 through com9. The debugger determines whether the S5U1C33000H or S5U1C330M1D1 (+ S5U1C330M2S) is connected.

`sim`: Simulator started (in SIM mode).

Time: Communication start delay time (in seconds, decimal, or hexadecimal)
After a COM port of the computer is opened, the debugger waits for the time set in seconds here before starting communication. When this entry is omitted, the debugger starts communication immediately. When operating in ICD3, ICD6, or simulator mode, this specification is unnecessary.

Usage example

■ Example 1

```
(gdb) target sim
boot () at ./boot.s:9
Connected to the simulator.
```

The debugger is set to simulator mode.

■ Example 2

```
(gdb) target icd usb
```

The debugger is set to ICD3 mode.

■ Example 3

```
(gdb) target icd com1 10
```

The debugger is set to ICD2 or debug monitor mode. After the COM1 port is opened, a wait time of 10 seconds is inserted before the debugger starts communicating with the S5U1C33000H or S5U1C330M1D1 (+ S5U1C330M2S).

Notes

- When you set a memory map by loading a parameter file, be sure to execute the `c33 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) c33 rpf sample.par     (sets map information.)
(gdb) file sample.elf        (loads debugging information.)
(gdb) target icd com1        (this command)
(gdb) load                   (loads the program.)
(gdb) c33 rsth               (hot reset)
```

- A COM port mounted on some personal computers may not be ready for use after being opened. Use the *Time* parameter to insert a wait time before the port becomes usable. This is especially important if you fail to connect your computer with the S5U1C33000H using a `target` command with *Time* omitted.

detach (disconnect target) [ICD2 / ICD3 / ICD6 / SIM / MON]

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
```

ICD3 mode is exited.

Notes

This command can be used to turn the S5U1C33000H or S5U1C33001H 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.

c33 das (debug unit address set) [ICD3 / ICD6]

Operation

Specifies the CPU type and debug unit addresses used to configure the ICD ver. 3/ver. 4/ver. 6 (S5U1C33001H).

Use this command only when the target uses the C33 PE Core and the debugging RAM base address is other than 0x71000.

The conditions specified using this command will be set to the ICD (S5U1C33001H) by the `target` command. Therefore, this command must be executed before the `target` command.

Format

`c33 das` ***DebugUnitBaseAddress DebugRAMAddress*** [***CPUtype***]

DebugUnitBaseAddress:	Debug unit start address (decimal or hexadecimal)
DebugRAMAddress:	Debug RAM start address (decimal or hexadecimal)
CPUtype:	CPU type **0** C33 STD (default) **1** C33 PE **3** S1C33401 (C33 ADV)

Usage example

```
(gdb) c33 das 0x60000 0x71000 3
(gdb) target icd usb
```

Specify 0x60000 for the debug ROM start address, 0x71000 for the debug RAM start address and C33 ADV Core for the CPU type using this command, then set these conditions into the ICD using the `target` command.

Notes

- The `c33 das` command is effective only in ICD3 and ICD6 modes.
- When only the `target` command is executed without this command, the conditions below are set into the ICD.

  Debug ROM start address: 0x60000
  Debug RAM start address: 0x71000
  CPU type: C33 STD

- Under normal conditions, use this command only once before the `target` command. However, if the debug unit's base address is changed thereafter, you must specify a new base address each time before re-executing this command. In such cases, the CPU type is ignored, so its specification may be omitted.
  Example:

```
(gdb) c33 das 0x60000 0x71000 1
(gdb) target icd usb
          Debugging launched
                  · · · Memory locations are changed in hardware in the middle of debugging.
(gdb) c33 das 0x800000 0x811000
          Debugging restarted
```

c33 lpt (parallel program load) [ICD2]

Function

Specifies whether high-speed download function by connecting between the PC and ICD ver. 2 (S5U1C33000H) with the parallel cable is used or not. This command must be executed before the `load` command. When this command is not executed, program is downloaded into the target via the serial (COM) port.

Format

`c33 lpt` [***PortNumber***]

PortNumber: Printer (LPT) port number on the PC (decimal or hexadecimal)
`1` or **`2`**
When omitted, the serial (COM) port is selected.

Usage example

```
(gdb) file test.elf
(gdb) target icd com1
(gdb) c33 lpt 1                  ← Sets parallel mode
(gdb) load
(gdb) c33 lpt                    ← Clears parallel mode
(gdb) load
```

The first `load` command performs high-speed downloading using the parallel port (LPT1).
The second `load` command performs normal downloading using the serial port.

Notes

- The `c33 lpt` command is effective only in ICD2 mode.
- This command must be executed before the `load` command.

pwd (display current directory)
cd (change current directory) [ICD2 / ICD3 / ICD6 / SIM / MON]

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/gnu33/sample/tst.
(gdb) cd /cygdrive/c/EPSON/gnu33/sample/asm
Working directory /cygdrive/c/EPSON/gnu33/sample/asm.
```

After the current directory is confirmed, it is changed to "`c:\EPSON\gnu33\sample\asm\`".

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.

c33 firmupdate (update firmware) [ICD3 / ICD6]

Function

Updates the firmware written in the flash memory on the ICD ver. 3/ver. 4/ver. 6 (S5U1C33001H). This command is effective only in an ICD mode (S5U1C33001H 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 (S5U1C33001H) off and on again.

Format

`c33 firmupdate` ***Filename***

Filename: Name of firmware file (Motorola S3 format file)

Usage example

```
(gdb) target icd usb
(gdb) c33 firmupdate icd33dmt.sa
```

Loads the firmware file `icd33dmt.sa` and updates the ICD (S5U1C33001H) firmware with the loaded contents.

Notes

The `c33 firmupdate` command is effective only in ICD3 and ICD6 modes.

c33 dclk (change DCLK cycle) [ICD2 / ICD3 / ICD6 / MON]

Operation

Sets the divide value to generate the DCLK clock from the core clock.

Format

`c33 dclk` ***Number***

Number:	Divide value 1, 2, 4, or 8

Usage example

■ Example 1

```
(gdb) c33 dclk 2
```

Sets the DCLK frequency to the half of the core clock.

■ Example 2

```
(gdb) c33 dclk 8
```

Sets the DCLK frequency to the 1/8 of the core clock.

Notes

This command cannot be used in simulator mode.

c33 oscwait (set wait counter in OSC1 mode) [ICD3 / ICD6]

Operation

When the system clock of the target processor is set to OSC1 (low-speed clock), the debugger may not execute or break program normally. This is because communication between the target and ICD may not be performed normally due to a reduction in the target operating speed.
This command adjusts the communication timing between the target and ICD in order that ICD can control the target normally even if the target operates with a low-speed clock.

Format

`c33 oscwait [Counter]`

Counter: Counter value for the software wait state inserted when starting or breaking a program execution (decimal or hexadecimal)
When omitted, counter is initialized to 0x30000.

Conditions: $0 \le Counter \le$ 0xffffff

Usage example

```
(gdb) c33 oscwait 0x40000
```

A wait state generated by executing a loop 0x40000 times is inserted when ICD starts or breaks executing the program.

Notes

- The `c33 oscwait` command is effective only in ICD3 and ICD6 modes.
- The initial counter value before setting with this command is 0x300. Leave this counter value unchanged except when the target operates with the OSC1 clock.
- Note that a larger count value decreases the response to execution of go/step/break commands.

c33 cachehit (display cache-hit rate) [ICD3 / ICD6]

Operation

Displays the cache-hit rate.
The hit rate is determined as "(1 - (mis-hit count/access count)) × 100%".

Format

```
c33 cachehit
```

Usage example

```
(gdb) c33 cachehit
The rate of cache hit is XX.XXXXX%
```

Notes

This command can be used only when the target is S1C33401 (C33 ADV Core model) and **gdb** is set in ICD3 and ICD6 modes.

c33 logiana (set logic analyzer mode) [ICD3 / ICD6]

Operation

This command allows use of the on-chip bus trace command (`c33 otd`, `c33 otf`) in user mode even if the target is a C33 STD or C33 PE Core model.

Format

`c33 logiana` ***Mode USER_IN***

Mode:	Logic analyzer mode enable	
	`On`	Enable
	`Off`	Disable
USER_IN:	**`0`**	Trace input (the external trace input is traced) (default)
	`1`	Trace trigger (the target CPU's trigger signal is traced)

Usage example

```
(gdb) c33 logiana on 1
logic analyzer mode on.
  target CPU trigger signal.
```

Enables the logic analyzer mode with trace trigger input.

Notes

This command cannot be used in the following cases:
- When the target is S1C33401 (C33 ADV Core model)
- When the debugger is set in ICD2, MON, or simulator mode

c33 help (help) [ICD2 / ICD3 / ICD6 / SIM / MON]

Operation

Displays a command description.

Format

`c33 help` [*Command*]
`c33 help` [*GroupNo.*]

Command: Name of command
GroupNo.: Command group number

Usage example

■ Example 1

```
(gdb) c33 help
group  0: memory ........... c33 fb,c33 fh,c33 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 ........ c33 rstc,c33 rsth,c33 rstt
group  4: interrupt ........ c33 int
group  5: break ............ break,tbreak,hbreak,thbreak,watch,rwatch,awatch, ···
group  6: symbol ........... info locals,info var,print
group  7: file ............. file,load,c33 lpt
group  8: map .............. c33 rpf,c33 map
group  9: flash memory ..... c33 fls,c33 fle
group 10: trace ............ c33 tm,c33 td,c33 ts,c33 tf,c33 obt,c33 otd,c33 otf
group 11: simulated I/O .... c33 stdin,c33 stdout
group 12: flash writer ..... c33 fwe,c33 fwlp,c33 fwld,c33 fwdc,c33 fwd
group 13: others ........... c33 dclk,c33 oscwait,c33 log,source,c33 clockmd, ···
 Please type "c33 help 1" to show group 1 or type "c33 help c33 fb" to get usage ···
```

When you omit parameters, a list of command groups is displayed.

■ Example 2

```
(gdb) c33 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
 c33 callmd         Set user function call mode
 c33 call           Call user function
 Please type "c33 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) c33 help step
step: Single-step,every line                  [ICD2/ICD3/ICD6/SIM/MON]

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) c33 help c33 rsth
c33 rsth: Hot reset                                [ICD2/ICD3/ICD6/SIM/MON]

usage: c33 rsth

example:
(gdb) c33 rsth
The CPU is hot-reset.
```

To display a C33 command, specify the command name including "`c33`".

Notes

- Executing the `help` command (that comes standard with gnu) instead of the `c33 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. [ICD2/ICD3/ICD6/SIM/MON]) appears in the usage display (see Examples 3 and 4) indicating the modes in which the command is effective.

  ICD2: The command can be used in ICD2 mode (when S5U1C33000H (ICD ver. 2) is used)
  ICD3: The command can be used in ICD3 mode (when S5U1C33001H1100 (ICD ver. 3) or S5U1C33001H 1200 (ICD ver. 4) is used)
  ICD6: The command can be used in ICD6 mode (when S5U1C33001H1400 (ICD ver. 6) is used)
  MON: The command can be used in debug monitor mode (when S5U1C330M1D1 and S5U1C330M2S is used)
  SIM: The command can be used in simulator mode (when debugging with the PC alone)

  If "[ICD2/MON]" is displayed, it indicates that the command cannot be executed in modes other than ICD2 or debug monitor mode.

c33 savewatch (saves symbols registered in Watch Window)

[ICD2/ICD3/ICD6/SIM/MON]

Operation

Saves symbols registered in the Watch Window in a file (CURRENT `PATH\watch.gdb`)
Symbols can be reregistered in the Watch Window by using the `c33 loadwatch` command.

Format

`c33` ***savewatch***

Usage example

When the following symbols are registered in the Watch Window

```
Name   Value
-----------------------------------------------------------------
main   {int ()} 0x12c <main>         //Function name
i      0x1234                        //int type variable
szBuf  unsigned long [10]            //Arrangement

The following symbol name is saved.
    main
    i
    szBuf
```

Notes

- Every time this command is executed, the contents stored in the .gdb file are updated.
- This command generates an error if the `target` command has not been executed previously.
  Error message: "`C33 command error, command is not supported in present mode.`"
- In the following cases, symbols are not saved.
  1. The Watch Window is closed.
  2. There is no registered symbol.
- The maximum number of symbols that can be stored is 300.

c33 loadwatch (reload saved symbols into Watch Window)

[ICD2/ICD3/ICD6/SIM/MON]

Operation

Reregisters symbols that were saved in the file (CURRENT `PATH\watch.gdb`) using the `c33 savewatch` command in the Watch Window.

Format

`c33` ***loadwatch***

Usage example

```
target sim
load
c33 loadwatch
```

Notes

- This command generates an error if the `target` command has not been executed previously.
  Error message: "`C33 command error, command is not supported in present mode.`"
- In the following cases, symbols are not registered.
  1. The Watch Window is closed.
  2. There is no symbol.
  3. Local symbols that cannot be handled by the functions of the current PC.
  4. The same symbols have already been registered.
  5. The `file` command has not been executed.
  6. There is no `watch.gdb` file in the current path.
- Nothing will happen if the `c33 savewatch` command has not been executed (`save breakpoints/watch` button not executed) or if the number of registered symbols is 0.
- The maximum number of symbols that can be registered is 300.

quit (quit debugger) [ICD2 / ICD3 / ICD6 / SIM / MON]

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)

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 `c33 rpf` command.

```
(gdb) c33 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, "GNU33 IDE".

Be sure to execute the `c33 rpf` command before the `file`, `target`, and `load` commands. The following shows the basic sequence of command execution:

```
(gdb) c33 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) c33 rsth              (hot reset)
```

How to create a parameter file

You can create parameter files by selecting [GNU33 Parameter Settings] from the [Properties] dialog box of the **IDE**. For details about **IDE**, see Chapter 5, "GNU33 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.

```
#gnu33 gdb parameter file                   (1)
C33_ADVANCED                                (2)

RAM      000000 001fff 00W    #IRAM         (3)
IO       040000 04ffff 00H    #IO
RAM      600000 6fffff 11H    #RAM
ROM      c00000 cfffff 55B  B #ROM
STACK    000000 001fff        #STACK        (4)
```

(1) Comment

From # to the end of the line is interpreted as a comment.

(2) CPU type

When debugging for a MCU model that incorporates the C33 ADV Core (S1C33401) or C33 PE Core, write C33_ADVANCED or C33_PE here, respectively. For MCU models incorporating the C33 STD Core, this specification is unnecessary.

This setting is only effective in simulator mode. In ICD2, ICD3, ICD6, and debug monitor modes, you need not specify this setting because the type of target CPU is automatically recognized. However, in order to debug a program in simulator mode, we recommend that you write C33_ADVANCED or C33_PE here when using the C33 ADV Core (S1C33401) or C33 PE Core.

(3) 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. In other connect modes, this parameter is ignored because said conditions are set in the actual target by the BCU registers.

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 other connect modes, 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 6fffff 11H   #RAM`	Effective
`ROM    600000 6fffff 22H   #ROM`	Ignored
`IO     600000 7fffff 33H   #I/O`	0x700000 through 0x7fffff 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.

(4) 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 ICD2, ICD3, ICD6, and debug monitor modes

- The debugger does not check the set addresses of software PC breaks.
- Trace data cannot be saved to a file by the `c33 tf` and `c33 otf` commands.

In simulator mode

- If the `target sim` command is executed without executing the `c33 rpf` command, simulated memory is reserved assuming that RAMs are mapped to the areas shown below. In this case, the initial value of memory is 0x00.

Addresses 0 to 0x1fff	(8KB)
Addresses 0x40000 to 0x4ffff	(64KB)
Addresses 0x600000 to 0x6fffff	(1MB)
Addresses 0xc00000 to 0xcfffff	(1MB)

- The set addresses of software PC breaks are not checked.
- 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 #, *function* at *file*:*line*	Made to break at a set breakpoint
Break by hard pc break1.	Made to break at hardware PC breakpoint 1
Break by hard pc break2.	Made to break at hardware PC breakpoint 2
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
Break by sleep or halt.	Made to break by executing `slp` or `halt` instruction in simulator mode
Illegal address exception.	Made to break by executing invalid address instruction in simulator mode
Illegal instruction.	Made to break by executing invalid instruction in simulator mode
Illegal delayed instruction.	Made to break by executing invalid delay instruction in simulator mode
Hardware read watchpoint #: *symbol*	Made to break for data read condition
Hardware access (read/write) watchpoint #: *symbol*	Made to break for data access (read/write) condition
Program received signal SIGTRAP, Trace/breakpoint trap.	Forcibly made to break by [Stop] button (in simulator mode)
Program received signal SIGINT, Interrupt.	Forcibly made to break by [Stop] button (in ICD mode)

10.9.2 Error Messages

Table 10.9.2.1 Error messages (in alphabetical order)

Message	Description
A setup of a serial port was not completed.	Serial communication rate with ICD is not 115200 or 38400 bps.
Address(0x*#*) is ext or delayed instruction.	The specified address cannot be set due to an `ext` or delayed instruction.
C33 command error, command is not supported in present mode.	The input command cannot be executed in current connection mode.
C33 command error, command is too long.	The input command exceeds 256 characters in length.
C33 command error, invalid command.	The command is erroneous.
C33 command error, number of parameter.	The number of command parameters is incorrect.
C33 command error, start address > end address.	The specified start address is greater than the end address.
C33 command error, start cycle < end cycle.	The specified start cycle is greater than the end cycle.
Cannot access memory at address	Cannot access address #.
Cannot allocate memory.	The necessary size of memory area as specified by a parameter could not be reserved.
Cannot clear data break(0x*#*).	The specified data break address is invalid; no breakpoints are set there.
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 open file(*file*).	Cannot open the file.
Cannot open ICD33 usb driver.	Failed to open the USB drive.
Cannot set data break any more.	Cannot set a data breakpoint.
Cannot set hard pc break any more.	The number of hardware PC breakpoints set exceeds the limit (up to 2).
Cannot set soft pc break any more.	The number of software PC breakpoints set exceeds the limit (up to 200).
Cannot write file.	Cannot write to the file.
Clock timer overflow.	The counter timed-out during clock measurement.
Communication error(bcc).	A BCC error occurred in the message received from ICD.
Communication error(host->ICD33).	Failed in USB transmission to ICD.
Communication internal error(*#*).	An internal error occurred while communicating with ICD.
Communication system error(*#*).	Connection was severed while communicating with ICD.
Framing error.	A framing error occurred during communication with ICD.
ICD33 is busy(*#*).	ICD is in a busy state.
Initialization error of ICD33.	Failed to initialize the target.

Message	Description
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 c33 architecture ELF file.	The file specified with the `file` command is not an elf format file supported in S5U1C33001C.
Load file is not motorola format.(*file*)	This is not a Motorola format file.
Load max address(0x*#*) overflow.	The address to be loaded exceeds the maximum value.
Load motorola file format error.(*file*)	The Motorola file to be loaded contains a format error.
Max address(0x*#*) overflow.	The specified address exceeds the maximum value.
No memory map information.	No map information can be found.
Receiving message is inaccurate.	A message exceeding the maximum size was received during communication with ICD.
Size limit(0x*#*) over.	The specified size exceeds the maximum value.
Size limit(0x*#*-0x*#*) over.	The specified size is out of the valid range.
Specification is required in the device for connecting.	The device name for ICD selection in the `target` command must be specified correctly.
Target connection causes communication error at *#* times.	An error occurred when checking SIO connection.
Target connection causes data error at *#* times.	Cannot perform SIO connection check.
Target connection causes time out error at *#* times.	SIO connection check timed-out.
There is no argument given to this command.	Failed to disconnect the target.
Timeout error #(ICD33 -> host).	Wait for reception from ICD timed-out during communication with ICD.
Transmitting failure(*#*).	NAK was received from ICD during communication with ICD.
Expression cannot be implemented with read/access watchpoint.	Cannot set data break for the specified address.
Please input "Editor path".	Enter characters in "Enter path".
"Editor path" exceeds 255 characters.	The number of characters entered in "Editor path" exceeds 255.
gnuEdit.gdb file save error	The external editor path could not be saved.

ICD denotes the S5U1C33000H or S5U1C33001H.

11 Other Tools

This chapter explains the other tools that are included in the S1C33 Family C/C++ Compiler Package.

11.1 make.exe

11.1.1 Functional Outline

The S1C33 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. 3.81), 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 S5U1C33001C only supports ordinary makes without target names and makes with the target name “`clean`” and a designation of “`all`”.

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 Gnu33 Plug-in for Eclipse
# This file should be placed directly under the project folder

# macro definitions for target file
TARGET= sample
GOAL= $(TARGET).elf

# macro definitions for tools
TOOL_DIR= C:/EPSON/GNU33
CC= $(TOOL_DIR)/xgcc
CXX= $(TOOL_DIR)/xgcc
AS= $(TOOL_DIR)/xgcc
AS_CC= $(TOOL_DIR)/as
LD= $(TOOL_DIR)/ld
RM= $(TOOL_DIR)/rm
SED= $(TOOL_DIR)/sed
CP= $(TOOL_DIR)/cp
OBJDUMP= $(TOOL_DIR)/objdump

# macro definitions for tool flags
CFLAGS= -B$(TOOL_DIR)/ -gstabs -S  -medda32  -O1 -I$(TOOL_DIR)/include -fno-builtin
-mlong-calls  -Wall
CXXFLAGS= ${CFLAGS}
ASFLAGS= -B$(TOOL_DIR)/ -c -xassembler-with-cpp -Wa,--gstabs  -Wa,-medda32
ASFLAGS_CC=  -medda32
LDFLAGS=  -Map sample.map -N -T sample_gnu33IDE.lds
ALLOBJDUMP_FILE= __allobjects
EXTFLAGS= -Wa,-mc33_ext -Wa,$(TARGET).dump -Wa,$(ALLOBJDUMP_FILE).dump
EXTFLAGS_CC= -mc33_ext $(TARGET).dump $(ALLOBJDUMP_FILE).dump
OBJDUMPFLAGS= -t

# macro for switching 2pass or 1pass build
PASS= 2pass

# search paths for source files
vpath %.c
vpath %.cpp
vpath %.cc
vpath %.cxx
vpath %.C
vpath %.s

# macro definitions for object files
OBJS= boot.o \
      main.o \

# macro definitions for library files
OBJLDS= $(TOOL_DIR)/lib/std/libstdio.a \
        $(TOOL_DIR)/lib/std/libstdc++.a \
        $(TOOL_DIR)/lib/std/libgcc2.a \
        $(TOOL_DIR)/lib/std/libc.a \
        $(TOOL_DIR)/lib/std/libgcc.a \
# macro definitions for assembly files generated from c source files
CEXTTEMPS= main.ext0 \

# macro definitions for dependency files
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
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
```

```
.PHONY : all clean

all : $(GOAL)

$(TARGET).elf : $(OBJS) sample_gnu33IDE.mak sample_gnu33IDE.lds
ifeq ($(PASS), 1pass)
# 1pass linking
  $(LD) $(LDFLAGS) -o $@ $(OBJS) $(OBJLDS)
else
# 1pass linking
  -$(LD) $(LDFLAGS) -o $@ $(OBJS) $(OBJLDS) 2>lderr
  @if [ -s lderr ]; then \
         cmd /c "type lderr" \
         && $(RM) -f $(TARGET).elf \
         && exit 1; \
  else $(RM) -f lderr ; \
  fi
  $(OBJDUMP) $(OBJDUMPFLAGS) $@ > $(TARGET).dump
  $(RM) -f $(TARGET).elf
# create all objets dump file
  $(RM) -f $(ALLOBJDUMP_FILE).dump
  for NAME in $(OBJS) ; do \
         $(OBJDUMP) $(OBJDUMPFLAGS) $$NAME >> $(ALLOBJDUMP_FILE).dump  ; done
# save 1pass object files
  @if [ -e obj1pass ]; then \
         cmd /c "rd /s /q obj1pass" ; \
  fi
  cmd /c "md obj1pass"
  for NAME in $(subst /,\\,$(OBJS)) ; do \
         cmd /c "copy /y $$NAME obj1pass\\$$NAME" >nul ; done \
  && $(RM) -f $(OBJS)
# 2pass for assembly files
  $(AS) $(ASFLAGS) $(EXTFLAGS) -o boot.o boot.s
# 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)
# restore 1pass object files
  $(RM) -f $(OBJS) \
  && \
  for NAME in $(subst /,\\,$(OBJS)) ; do \
         cmd /c "copy /y obj1pass\\$$NAME $$NAME" >nul ; done \
  && cmd /c "rd /s /q obj1pass"
endif
  @cmd /c "echo ---------------- Finished building target : $@ ----------------"

## boot.s
boot.o : boot.s
  $(AS) $(ASFLAGS) -o $@ $<
  $(DEPCMD_AS)

## main.c
main.o : main.c main.ext0
  $(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<
  $(AS_CC) $(ASFLAGS_CC) -o $@ $(@:%.o=%.ext0)
  $(DEPCMD_CC)

# dependecies for assembled c source files
main.ext0 : main.c

# include dependency files
-include $(DEPS)

# clean files
clean :
  $(RM) -f $(OBJS) $(TARGET).elf $(TARGET).map $(DEPS) $(CEXTTEMPS) $(TARGET).dump
$(ALLOBJDUMP_FILE).dump lderr
  @if [ -e obj1pass ]; then \
         cmd /c "rd /s /q obj1pass" ; \
  fi
```

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/gnu33/sample/tst/boot.o : c:/EPSON/gnu33/sample/tst/boot.s
       as -o boot.o c:/EPSON/gnu33/sample/tst/boot.s
                         ↓
/cygdrive/c/EPSON/gnu33/sample/tst/boot.o : /cygdrive/c/EPSON/gnu33/sample/tst/boot.s
       as -o boot.o c:/EPSON/gnu33/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/gnu33
       :
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/gnu33/ld -T sample.lds -Map sample.map -N -o sample.elf boot.o main.o
/cygdrive/c/EPSON/gnu33/lib/libc.a /cygdrive/c/EPSON/gnu33/lib/libgcc.a
```

Predefined macros

`$@` used in the example above is a predefined macro. The following three predefined macros are available. 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...)
```

$< This will be replaced with the first dependency file (including the extension) of the target currently being processed.

Example:

```
main.o : main.c ...
    xgcc -o $@ $< (= xgcc -o main.o main.c)
```

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.

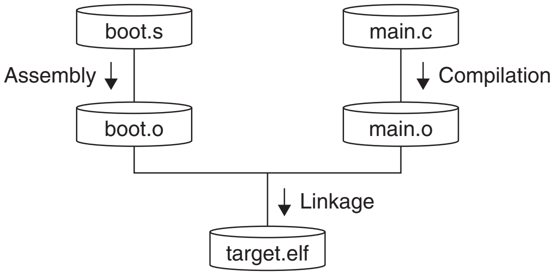


Figure 11.1.6.1 Relationship of files (example)

Sample of a make file:

```
target.elf : boot.o main.o                                        (A)
     $(LD) $(LDFLAGS) -o target.elf boot.o main.o $(LIBS)

boot.o : boot.s                                                   (B)
     $(AS) $(ASFLAGS) -o boot.o boot.s

main.o : main.c                                                   (C)
     $(CC) $(CFLAGS) main.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/gnu33/sed.exe -f ../comm/place.sed vector.c > vector2.c
     /cygdrive/c/EPSON/gnu33/xgcc -B/cygdrive/c/EPSON/gnu33/ -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/gnu33/xgcc -B/cygdrive/c/EPSON/gnu33/ -c vector2.c -o vector2.o

  vector2.c : vector.c
     /cygdrive/c/EPSON/gnu33/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
```

Suffix definition

```
.c.o :
  $(CC) $(CFLAGS) -o $(SRC_DIR)/$*.o $(SRC_DIR)/$*.c
.s.o :
  $(AS) $(ASFLAGS) -o $(SRC_DIR)/$*.o $(SRC_DIR)/$*.s
```

Suffix rules

```
# 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
```

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/gnu33
   :
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 Invocation by sh.exe

If the command line in a makefile contains a shell metacharacter such as a semicolon (;), a switching symbol (<, >,>>, |), a replacement symbol (*, ?, [], $, =), a quote mark, an escape character, or a comment (““, ‘, `, \, #, etc., :), the make will launch the Bourne shell and process the command.

If there is no need for syntax analysis of the command line by a shell, the make executes the command directly. In this case, the Bourne shell used is sh.exe at the location where make.exe is found.

11.1.10 Messages

The following shows the messages generated by the **make.exe**:

Table 11.1.10.1 Normal message

Message	Description
make: Nothing to be done for '*TARGET*'.	No process has been executed for creating the *TARGET*. This error will occur if no command is defined for creating a target file.
make: '*FILENAME*' is up to date.	*FILENAME* has already been updated. The make process is terminated without executing a command.

Table 11.1.10.2 Error messages

Error message	Description
make: *** No rule to make target '*FILENAME1*', needed by '*FILENAME2*'. Stop.	*FILENAME1* required for generating *FILENAME2* cannot be found, or a dependency list for generating *FILENAME1* has not been defined. The make process is terminated.
make: *TOOL*: Command not found	*TOOL* specified in the dependency list for generating *FILENAME* cannot be found. The make process is terminated.
make: *** [*FILENAME*] Error 127	
make: *** [*FILENAME*] Error 1	An error has occurred in the tool invoked. The make process is terminated.
XXX.mak:*LINE*: *** missing separator. Stop.	There is an illegal separator symbol at the line *LINE* in the *XXX*`.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.
XXX.mak:*LINE*: *** commands commence before first target. Stop.	A command was found before the first target definition in the line *LINE* in the *XXX*.mak file. The make process was aborted. This error will occur if TAB is specified before the first target.(like “all”).

Table 11.1.10.3 Warning messages

Warning message	Description
make: *** Warning: File '*FILENAME*' has modification time in the future (*yyyy*-*mm*-*dd* *hh*:*mm*:*ss* > *yyyy*-*mm*-*dd* *hh*:*mm*:*ss*)	The time stamp of *FILENAME* 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.	

11.1.11 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\gnu33\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.

`-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\gnu33\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\gnu33\>objdump -d main.o
main.o:     file format elf32-c33

Disassembly of section .text:

00000000 <main>:
   0: 0200  pushn   %r0             pushn    %r0

00000002 <.LBB2>:
   2: 6c04  ld.w    %r4,0x0         ld.w     %r4,0x0
   4: c000  ext     0x0
   6: c000  ext     0x0
   8: 3cf4  ld.w    [%r15],%r4      xld.w     [%r15+0x0],%r4
.....
```

`-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\gnu33\>objdump -h test.elf

test.elf:     file format elf32-c33

Sections:
Idx Name           Size      VMA       LMA       File off  Algn
  0 .bss           00000004  00000000  00000000  00000094  2**2
                   ALLOC
  1 .data          00000008  00000004  00c000c4  00000158  2**2
                   CONTENTS, ALLOC, LOAD, DATA
  2 .text          000000bc  00c00000  00c00000  00000094  2**2
                   CONTENTS, ALLOC, LOAD, CODE
  3 .rodata        00000008  00c000bc  00c000bc  00000150  2**2
                   CONTENTS, ALLOC, LOAD, DATA
  4 .stab          00000288  00c000c4  00c000c4  00000160  2**2
                   CONTENTS, READONLY, DEBUGGING
  5 .stabstr       00000338  00c0034c  00c0034c  000003e8  2**0
                   CONTENTS, READONLY, DEBUGGING
  6 .comment       0000004c  00c00684  00c00684  00000720  2**0
                   CONTENTS, READONLY

*** 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\gnu33\>objdump -g test.elf

test.elf:     file format elf32-c33

boot.s:
/* file boot.s line 9 addr 0xc00004 */
/* file boot.s line 10 addr 0xc00008 */
/* file boot.s line 11 addr 0xc0000a */
/* file boot.s line 12 addr 0xc00010 */
/* file boot.s line 13 addr 0xc00016 */
./main.c:
typedef int32 int;
typedef int8 char;
typedef int32 long int;
typedef uint32 unsigned int;
.....
typedef void void;
long int aaaa /* 0xc000bc */;
long int bbbb /* 0x4 */;
long int aaaa2 /* 0xc000c0 */;
long int bbbb2 /* 0x8 */;
int main ()
{ /* 0xc0001c */
  /* file ./main.c line 12 addr 0xc0001c */
  { /* 0xc0001e */
    register int j /* 0x0 */;
    /* file ./main.c line 13 addr 0xc0001e */
    /* file ./main.c line 15 addr 0xc0001e */
    /* file ./main.c line 16 addr 0xc00026 */
    /* file ./main.c line 18 addr 0xc00028 */
    /* file ./main.c line 19 addr 0xc00030 */
    /* file ./main.c line 16 addr 0xc00030 */
  } /* 0xc00034 */
  /* file ./main.c line 20 addr 0xc00034 */
} /* 0xc00038 */
.....
int i /* 0x0 */;
```

`-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\gnu33\>objdump -t test.elf

test.elf:     file format elf32-c33

SYMBOL TABLE:
00000000 l    d  .bss   00000000
00000004 l    d  .data  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\gnu33\>objdump -s test.elf

test.elf:     file format elf32-c33

Contents of section .data:
 0004 ffffffff 7f7f7f7f                      ........
Contents of section .text:
 c00000 0400c000 20c00f6c f1a000c0 00c00f6c  .... ..l.......l
 c00010 00c000c0 041cf8df ffdff51e 0002046c  ...............l
 c00020 00c000c0 f43c402e 00c000c0 061d062e  .....<@.........
 c00030 fc1f1060 40024006 16700818 00c000c0  ...`@.@..p......
 c00040 f4301460 00c000c0 f43c4006 0302622e  .0.`.....<@...b.
 c00050 732e802e 912e272e ffc3f86f 762e8632  s.....'....ov..2
 c00060 87888788 042e452e 85328488 848856aa  ......E..2....V.
.....
```

`-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.

```
Contents of .stab section:

Symnum n_type n_othr n_desc n_value  n_strx String

-1     HdrSym 0      53     00000338 1
0      SO     0      0      00c00000 1      boot.s
1      SLINE  0      9      00c00004 0
2      SLINE  0      10     00c00008 0
3      SLINE  0      11     00c0000a 0
4      SLINE  0      12     00c00010 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\gnu33\>objdump -S test.elf

test.elf:     file format elf32-c33

Disassembly of section .text:

00c00000 <__START_text>:
  c00000: 0004  nop                   ***
  c00002: 00c0  pops    %psr          ***

00c00004 <boot>:
    .align 2
    .global boot
    .long   boot
boot:
    xld.w   %r15,0x0800
  c00004: c020  ext     0x20
  c00006: 6c0f  ld.w    %r15,0x0        xld.w     %r15,0x800
    ld.w    %sp,%r15    ; set SP
  c00008: a0f1  ld.w    %sp,%r15        ld.w     %sp,%r15
    xld.w   %r15,__dp   ; set default data area pointer
.....
00c0001c <main>:
const long aaaa2 =0x87654321;
long bbbb2 =0x7f7f7f7f;

main()
    {
  c0001c: 0200  pushn   %r0             pushn    %r0

00c0001e <.LBB2>:
    int j;

    i = 0;
  c0001e: 6c04  ld.w    %r4,0x0         ld.w     %r4,0x0
  c00020: c000  ext     0x0
  c00022: c000  ext     0x0
  c00024: 3cf4  ld.w    [%r15],%r4      xld.w     [%r15+0x0],%r4

00c00026 <.LM4>:
    for (j=0 ; ; j++)
  c00026: 2e40  ld.w    %r0,%r4         ld.w     %r0,%r4
```

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 (*filename*) 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.
- Note that the `ald` instruction cannot be disassembled if you execute objdump with the `-d` option to disassemble a "`.o`" or elf file that has been built with the `-mc33adv` option (for S1C33401) specified.

  Example:
  ```
  ext     0x0
  ld.w    %r1,0x30
  ld.w    %r1,[%r1]
  ```

  These instructions are disassembled as follows:
  ```
  xld.w   %r1
  ld.w    %r1,[%r1]
  ```

  Cannot be disassembled as follows:
  ```
  ald.w   %r1,0x30
  ```

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 S1C33 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 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 S1C33 Family:

`-I elf32-little`
Function: **Specify the input file format**
Explanation: Specifies elf as the input file format.

`-O srec`
Function: **Output in Motorola format**
Explanation: Specifies the Motorola format as the output file format. This option must be specified together with '`-I elf32-little`'.

`-O binary`
Function: **Output in binary format**
Explanation: Specifies binary format as the output file format. This option must be specified together with '`-I elf32-little`'.

`--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. This option must be specified together with '`-I elf32-little`'.

`-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\gnu33\>objcopy -I elf32-little -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\gnu33\>objcopy -I elf32-little -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`)

`func5.0` 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 S1C33 Family, the **moto2ff** is used to convert a HEX file generated by the **objcopy** into mask ROM data to be submitted to Seiko Epson.

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

Mask ROM data file

File format: Motorola S3 format file in which the unused area is filled with 0xff
File name: <*file name*>`.saf`
Description: A mask ROM data file. Submit this file to Seiko Epson.

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.
- 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 "`*FILENAME*`".`	The input file *FILENAME* cannot be opened.
`Cannot open output file "`*FILENAME*`".`	The output file *FILENAME* 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
`"`*FILENAME*`" contains data outside of specified range ("`*STARTADDR*`":"`*SIZE*`")`	The input file *FILENAME* contains data that exceeds the range specified by a start address and a block size. Data exceeded the specified range is not output.
`Invalid file format in "`*FILENAME*`" line "`*NUMBER*`".`	The input file *FILENAME* contains an invalid format data at line *NUMBER*.

11.6.5 Creating Mask ROM Data

After a Motorola S3 format HEX file has been generated by the **objcopy**, convert it into mask data using the **moto2ff**.

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

Example: `C:\EPSON\gnu33\>moto2ff 600000 100000 input.sa`

The command above outputs the data of 0x100000 bytes starting from address 0x600000 contained in `input.sa` to `input.saf`. The unused addresses within the range from addresses 0x600000 to 0x6fffff are filled with 0xff.

After creating a mask ROM file for the internal ROM according to the above procedure, be sure to perform the final verification of program operation using that file.

After verification, rename the mask ROM file to the one specified by Seiko Epson before presenting it to Seiko Epson.

How to generate two mask data files from a HEX file

Although a warning message will be displayed if `input.sa` contains data ranging outside of 0x600000 to 0x6fffff, the output file will be generated.

```
Warning: input.sa contains data outside of specified range (600000:100000)
```

Do the following procedure to generate two mask data files from a HEX file.

Example: When the HEX file contains data starting from 0x600000 and from 0xc00000

1. Create a copy of `input.sa` and name it `input2.sa`.
2. Execute **moto2ff** with `input.sa`.
   ```
   C:\EPSON\gnu33\>moto2ff 600000 100000 input.sa
   ```
3. Execute **moto2ff** with `input2.sa`.
   ```
   C:\EPSON\gnu33\>moto2ff C00000 100000 input2.sa
   ```

Ignore the warning messages appeared in Steps 2 and 3. The output files are generated normally.

By the above procedure, the mask data files `input.saf` containing data from 0x600000 and `input2.saf` containing data from 0xc00000 are generated.

THIS PAGE IS BLANK.

S1C33 Family C/C++ Compiler Package

Quick Reference

EPSON

CMOS 32-bit Single Chip Microcomputer

S1C33 Family C/C++ Compiler Package

Quick Reference for Development

Memory Map and Vector Table (C33 STD Core)

S1C33000 Core CPU

Memory Map

Address	Area		Area size
0xFFFFFFF	Area 18	External memory	64MB
	Area 17	External memory	64MB
	Area 16	External memory	32MB
	Area 15	External memory	32MB
	Area 14	External memory	16MB
	Area 13	External memory	16MB
	Area 12	External memory	8MB
0x1000000	Area 11	External memory	8MB
0x0C00000	Area 10	External memory	4MB
	Area 9	External memory	4MB
	Area 8	External memory	2MB
	Area 7	External memory	2MB
	Area 6	External I/O	1MB
	Area 5	External memory	1MB
0x0100000	Area 4	External memory	1MB
0x0080000	Area 3	On-chip ROM	512KB
0x0060000	Area 2	Reserved	128KB
0x0030000	Area 1	Internal I/O	128KB
0x0000000	Area 0	On-chip RAM	256KB

Vector Table

	Vector address
Reset	base + 0
Reserved	base + 4–12
Zero division	base + 16
Reserved	base + 20
Address error	base + 24
NMI	base + 28
Reserved	base + 32–44
Software exception 0 : Software exception 3	base + 48 : base + 60
External maskable interrupt 0 : External maskable interrupt 215	base + 64 : base + 924

base: Vector table start address
= 0x0080000 (when booting by on-chip ROM)
= 0x0C00000 (when booting by external ROM)

Registers (C33 STD Core)

S1C33000 Core CPU

General-purpose Registers (16)

31 0
R15
R14
R13
:
R4
R3
R2
R1
R0

Special Registers (5)

31 0	
PC	Program counter
PSR	Processor status register
SP	Stack pointer
ALR	Arithmetic operation low register
AHR	Arithmetic operation high register

(AHR, ALR: Option for Multiplication & Accumulation, Multiplication, and Division)

PSR

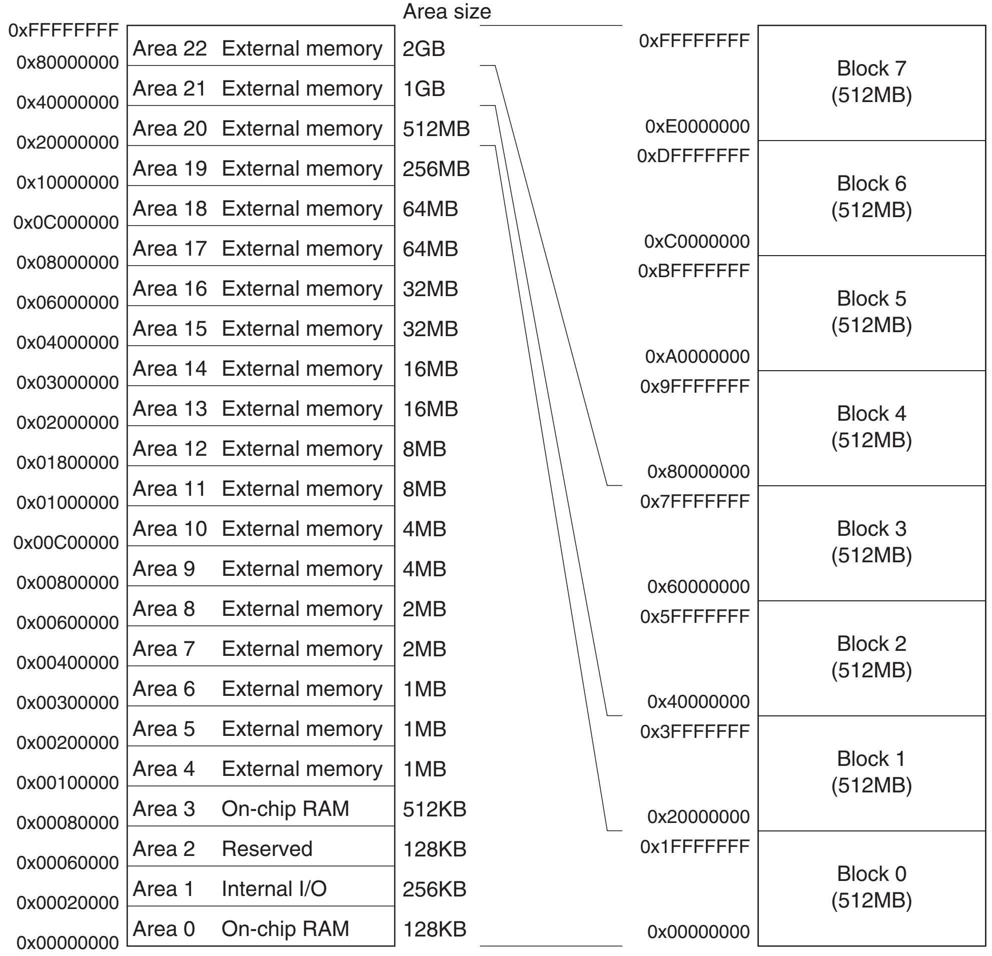
31–12	11–8	7	6	5	4	3	2	1	0
Reserved	IL	MO	DS	–	IE	C	V	Z	N

IL:	Interrupt level	(0–15: Enabled interrupt level)
MO:	MAC overflow flag	(1: MAC overflow, 0: Not overflown)
DS:	Dividend sign flag	(1: Negative, 0: Positive)
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)

Memory Map and Vector Table (C33 ADV Core)

Memory Map (Physical Address)

Memory Map (MMU Logical Address)



			Area size
0xFFFFFFFF			
0x80000000	Area 22	External memory	2GB
0x40000000	Area 21	External memory	1GB
0x20000000	Area 20	External memory	512MB
0x10000000	Area 19	External memory	256MB
0x0C000000	Area 18	External memory	64MB
0x08000000	Area 17	External memory	64MB
0x06000000	Area 16	External memory	32MB
0x04000000	Area 15	External memory	32MB
0x03000000	Area 14	External memory	16MB
0x02000000	Area 13	External memory	16MB
0x01800000	Area 12	External memory	8MB
0x01000000	Area 11	External memory	8MB
0x00C00000	Area 10	External memory	4MB
0x00800000	Area 9	External memory	4MB
0x00600000	Area 8	External memory	2MB
0x00400000	Area 7	External memory	2MB
0x00300000	Area 6	External memory	1MB
0x00200000	Area 5	External memory	1MB
0x00100000	Area 4	External memory	1MB
0x00080000	Area 3	On-chip RAM	512KB
0x00060000	Area 2	Reserved	128KB
0x00020000	Area 1	Internal I/O	256KB
0x00000000	Area 0	On-chip RAM	128KB

0xFFFFFFFF 0xE0000000	Block 7 (512MB)
0xDFFFFFFF 0xC0000000	Block 6 (512MB)
0xBFFFFFFF 0xA0000000	Block 5 (512MB)
0x9FFFFFFF 0x80000000	Block 4 (512MB)
0x7FFFFFFF 0x60000000	Block 3 (512MB)
0x5FFFFFFF 0x40000000	Block 2 (512MB)
0x3FFFFFFF 0x20000000	Block 1 (512MB)
0x1FFFFFFF 0x00000000	Block 0 (512MB)

Vector Table

	Vector address
Reset	TTBR + 0
Reserved	TTBR + 4–12
Zero division	TTBR + 16
Reserved	TTBR + 20
Address error	TTBR + 24
NMI	TTBR + 28
Reserved	TTBR + 32–44
Software exception 0 : Software exception 3	TTBR + 48 : TTBR + 60
External maskable interrupt 0 : External maskable interrupt 239	TTBR + 64 : TTBR + 1020

TTBR: Vector table start address
= 0x20000000 (initial value)

Registers (C33 ADV Core)

General-purpose Registers (16)

31 0
R15
R14
R13
:
R6
R5(AHR)
R4(ALR)
R3
R2
R1
R0

Special Registers (13)

31 0	
PC	Program counter (R only)
SSP	Supervisor stack pointer*1
USP	Users stack pointer
DP	Data pointer
TTBR	Vector table base register*1
SOR	Shift out register
LEA	Loop end address

31 0	
LSA	Loop start address
LCO	Loop counter
AHR	Arithmetic operation high register
ALR	Arithmetic operation low register
SP	Stack pointer
PSR	Processor status register
IDIR	Processor identification register (R only)
DBBR	Debug base register (R only)

*1 Read only in user mode

PSR

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	HE	RM	LM	PM	RC[3:0]				–	SW	OC	SE	–	–	LC	HC
Initial value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Supervisor mode	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r	r/w	r/w	r/w	r	r	r/w	r/w
User mode	r	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r	r/w	r/w	r/w	r	r	r/w	r/w

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	S	DE	ME	SV	IL[3:0]				MO	DS	–	IE	C	V	Z	N
Initial value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Supervisor mode	r/w	r	r	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r	r/w	r/w	r/w	r/w	r/w
User mode	r/w	r	r	r	r	r	r	r	r/w	r/w	r	r	r/w	r/w	r/w	r/w

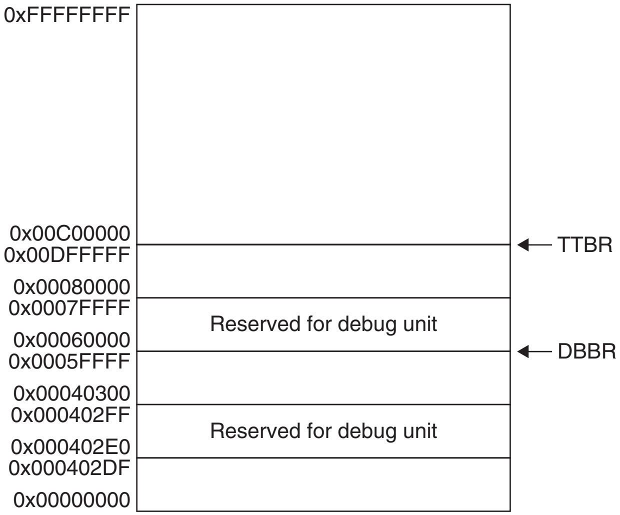
HE:	HALT/SLEEP enable	(1: halt/slp instruction enabled, 0: Disabled)
RM:	Repeat mode enable	(1: Repeat mode, 0: Normal)
LM:	Loop mode enable	(1: Loop mode, 0: Normal)
PM:	PUSH/POP mode	(1: Push/pop mode, 0: Normal)
RC:	Register counter	(0–15: push/pop start register number)
SW:	Scan word enable	(1: 32 bits, 0: 8 bits)
OC:	Overflow clear enable	(1: V clear enabled, 0: Disabled)
SE:	Shift with carry enable	(1: With carry, 0: Disabled)
LC:	ALR change enable	(1: ALR→R4, 0: Disabled)
HC:	AHR change enable	(1: AHR→R5, 0: Disabled)

S:	Saturation	(1: Saturation occurred, 0: Not occurred)
DE:	Debugging exception	(1: Exception occurred, 0: Not occurred)
ME:	MMU exception	(1: Exception occurred, 0: Not occurred)
SV:	Supervisor mode	(1: User mode, 0: Supervisor mode)
IL:	Interrupt level	(0–15: Enabled interrupt level)
MO:	MAC overflow flag	(1: MAC overflow, 0: Not overflown)
DS:	Dividend sign flag	(1: Negative, 0: Positive)
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)

Memory Map and Vector Table (C33 PE Core)

S1C33000 Core CPU

Memory Map


0xFFFFFFFF
0x00C00000
TTBR
0x00DFFFFF
0x00080000
0x0007FFFF
Reserved for debug unit
0x00060000
DBBR
0x0005FFFF
0x00040300
0x000402FF
Reserved for debug unit
0x000402E0
0x000402DF
0x00000000

Vector Table

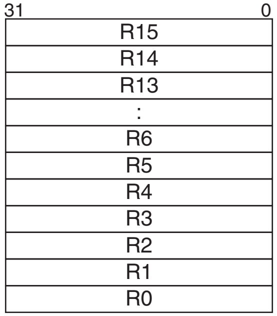
	Vector address
Reset	TTBR + 0
Reserved	TTBR + 4
`ext` exception	TTBR + 8
Undefined instruction exception	TTBR + 12
Reserved	TTBR + 16~20
Address error	TTBR + 24
NMI	TTBR + 28
Reserved	TTBR + 32~44
Software exception 0 : Software exception 3	TTBR + 48 : TTBR + 60
External maskable interrupt 0 : External maskable interrupt 239	TTBR + 64 : TTBR + 1020

TTBR: Vector table start address
= 0xC00000 (initial value)

Registers (C33 PE Core)

S1C33000 Core CPU

General-purpose Registers (16)


31
0
R15
R14
R13
:
R6
R5
R4
R3
R2
R1
R0

Special Registers (8)

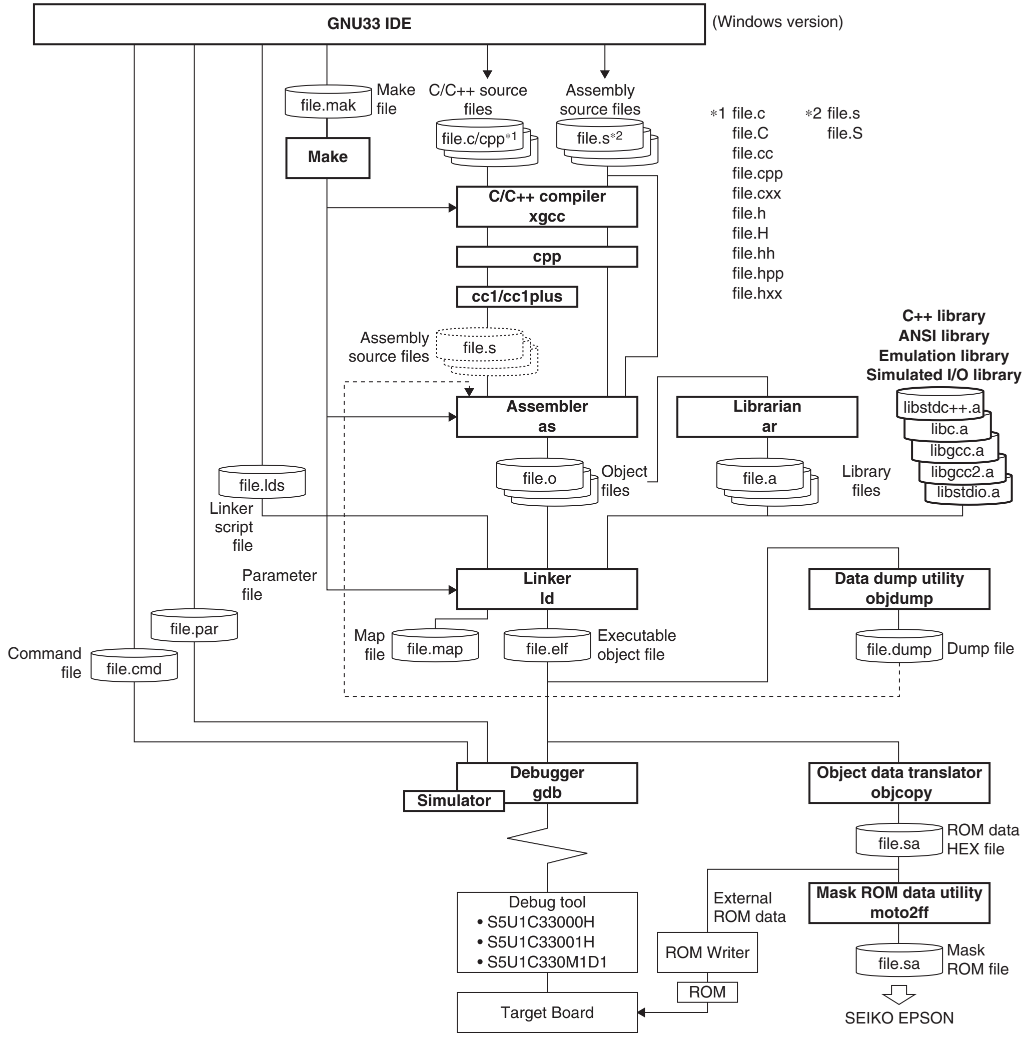
Register (31–0)	
PC	Program counter (R only)
DBBR	Debug base register (R only)
IDIR	Processor identification register (R only)
TTBR	Vector table base register
AHR	Arithmetic operation high register
ALR	Arithmetic operation low register
SP	Stack pointer
PSR	Processor status register

PSR

31–12	11–8	7	6	5	4	3	2	1	0
Reserved	IL	–	–	–	IE	C	V	Z	N

IL:	Interrupt level	(0–15: 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


GNU33 IDE
(Windows version)
file.mak
Make file
Make
C/C++ source files
file.c/cpp*1
Assembly source files
file.s*2
*1 file.c
file.C
file.cc
file.cpp
file.cxx
file.h
file.H
file.hh
file.hpp
file.hxx
*2 file.s
file.S
C/C++ compiler
xgcc
cpp
cc1/cc1plus
Assembly source files
file.s
Assembler
as
file.o
Object files
Librarian
ar
file.a
Library files
C++ library
ANSI library
Emulation library
Simulated I/O library
libstdc++.a
libc.a
libgcc.a
libgcc2.a
libstdio.a
file.lds
Linker script file
Parameter file
Linker
ld
Data dump utility
objdump
file.par
Map file
file.map
file.elf
Executable object file
file.dump
Dump file
Command file
file.cmd
Debugger
gdb
Simulator
Object data translator
objcopy
file.sa
ROM data HEX file
External ROM data
Mask ROM data utility
moto2ff
Debug tool
• S5U1C33000H
• S5U1C33001H
• S5U1C330M1D1
ROM Writer
ROM
file.sa
Mask ROM file
Target Board
SEIKO EPSON

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/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 mask ROM data file using the **objcopy** and the **moto2ff**.
   5-2) Submit the created data file to Seiko Epson.

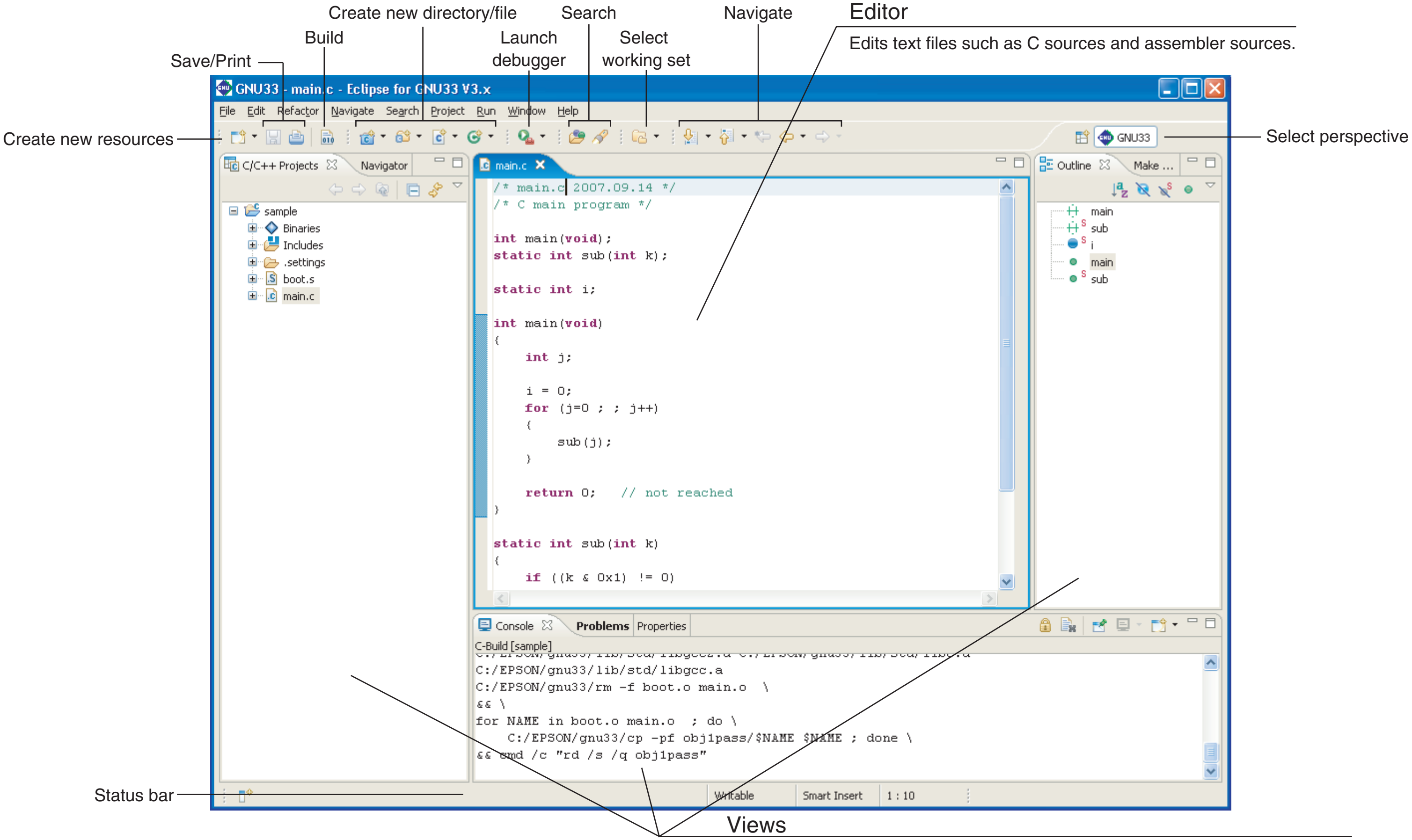
Overview

eclipse.exe

The **GNU33 IDE** provides an integrated development environment that allows the user to easily develop software with the S1C33 Family C/C++ Compiler Package (S5U1C33001C).

Start-up Command

`eclipse`


Create new directory/file
Search
Navigate
Build
Launch debugger
Select working set
Save/Print
Create new resources
Select perspective
Status bar
GNU33 - main.c - Eclipse for GNU33 V3.x
File Edit Refactor Navigate Search Project Run Window Help
GNU33
C/C++ Projects
Navigator
sample
Binaries
Includes
.settings
boot.s
main.c
Outline
Make ...
main
sub
i
main
sub
Console
Problems
Properties
C-Build [sample]
Writable
Smart Insert
1 : 10

```
/* main.c 2007.09.14 */
/* C main program */

int main(void);
static int sub(int k);

static int i;

int main(void)
{
    int j;

    i = 0;
    for (j=0 ; ; j++)
    {
        sub(j);
    }

    return 0;   // not reached
}

static int sub(int k)
{
    if ((k & 0x1) != 0)
```

```
C:/EPSON/gnu33/lib/std/libgcc.a
C:/EPSON/gnu33/rm -f boot.o main.o  \
&& \
for NAME in boot.o main.o  ; do \
    C:/EPSON/gnu33/cp -pf obj1pass/$NAME $NAME ; done \
&& cmd /c "rd /s /q obj1pass"
```

Editor

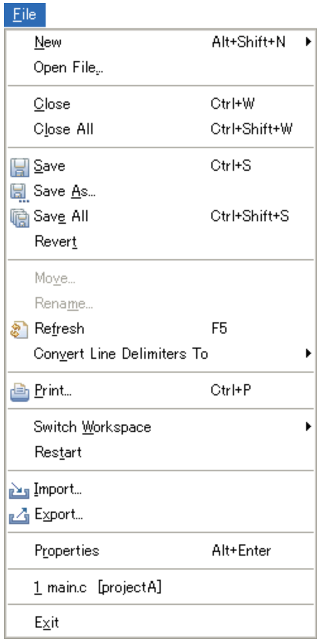
Edits text files such as C sources and assembler sources.

Views

Displays various information by contents. To open a closed view, select it from [Show View] on the [Window] menu.

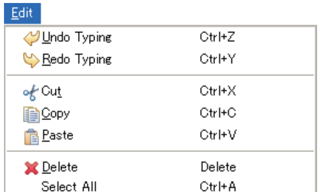
Menu Bar

[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...
Refresh	F5
Convert Line Delimiters To
Print...	Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties	Alt+Enter
1 main.c [projectA]
Exit

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.
Restart Restarts the IDE.
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**.

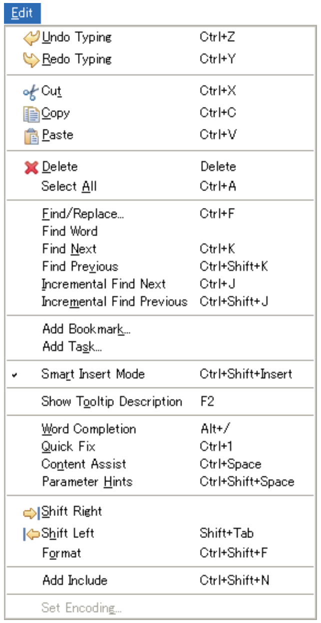
[Edit] menu


Edit
Undo Typing	Ctrl+Z
Redo Typing	Ctrl+Y
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Delete	Delete
Select All	Ctrl+A

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.

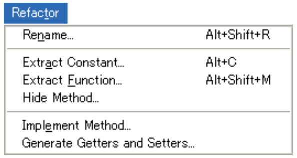
Menu Bar

[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 Word
Find Next	Ctrl+K
Find Previous	Ctrl+Shift+K
Incremental Find Next	Ctrl+J
Incremental Find Previous	Ctrl+Shift+J
Add Bookmark...
Add Task...
✓ Smart Insert Mode	Ctrl+Shift+Insert
Show Tooltip Description	F2
Word Completion	Alt+/
Quick Fix	Ctrl+1
Content Assist	Ctrl+Space
Parameter Hints	Ctrl+Shift+Space
Shift Right
Shift Left	Shift+Tab
Format	Ctrl+Shift+F
Add Include	Ctrl+Shift+N
Set Encoding...

Find/Replace... (Ctrl+F) Finds and replaces a string in the editor.
Find word Searches for the next occurrence matching the search string.
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).
Smart Insert Mode (Ctrl+Shift+Insert) Changes the editor Smart Insert Mode.
Show Tooltip Description (F2) Press the [F2] key while a tooltip is displayed to focus on the tooltip.
Word Completion (Alt+/) Inserts a word beginning with the character being entered in the editor at the current position.
Quick Fix (Ctrl+1) Displays proposals for error/warning corrections.
Content Assist (Ctrl+Space) Helps enter a C source keyword or template.
Parameter Hints (Ctrl+Shift+Space) Displays a tip for function arguments.
Shift Right Moves the beginning of a line to the right.
Shift Left (Shift+Tab) Moves the beginning of a line to the left.
Format (Ctrl+Shift+F) Formats a text according to formatter settings.
Set Encoding... Selects the text-encoding format.

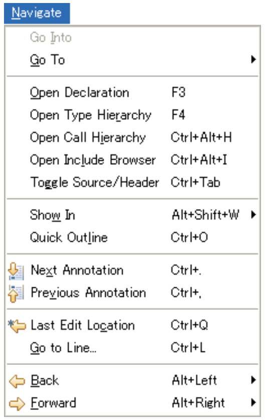
[Refactor] menu


Refactor
Rename...	Alt+Shift+R
Extract Constant...	Alt+C
Extract Function...	Alt+Shift+M
Hide Method...
Implement Method...
Generate Getters and Setters...

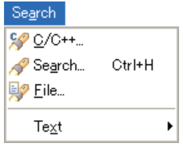
Rename... (Alt+Shift+R) Changes the name of a selected function or variable.
Extract Constant (Alt+C) Cuts out a constant from a source file for use in a variable.
Extract Function (Alt+Shift+M) Cuts out a part of a code from a source file for use in a function.

Menu Bar

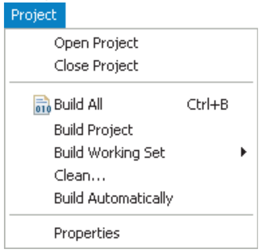
[Navigate] menu


Navigate
Go Into
Go To
Open Declaration F3
Open Type Hierarchy F4
Open Call Hierarchy Ctrl+Alt+H
Open Include Browser Ctrl+Alt+I
Toggle Source/Header Ctrl+Tab
Show In Alt+Shift+W
Quick Outline Ctrl+O
Next Annotation Ctrl+.
Previous Annotation Ctrl+,
Last Edit Location Ctrl+Q
Go to Line... Ctrl+L
Back Alt+Left
Forward Alt+Right

[Search] menu


Search
C/C++...
Search... Ctrl+H
File...
Text

[Project] menu


Project
Open Project
Close Project
Build All Ctrl+B
Build Project
Build Working Set
Clean...
Build Automatically
Properties

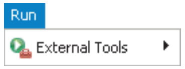
Go Into Changes display of the [C/C++ Projects]/[Navigator] view to display the content of just the currently selected directory.
Go To Navigates the display history of the [C/C++ Projects]/[Navigator] view.
Open Declaration (F3) Opens the declaration or definition of a selected object.
Open Type Hierarchy (F4) Opens the type hierarchy of a selected variable.
Open Call Hierarchy (Ctrl + Alt + H) Opens the call hierarchy of a selected function.
Open Include Browser (Ctrl + Alt + I) Opens the include hierarchy of a selected source file.
Toggle Source/Header (Ctrl + Tab) Switches to the corresponding source file and header file with the editor.
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... (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.

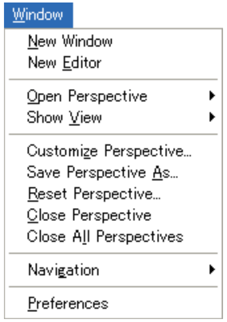
Open Project Opens the closed project currently selected in the [C/C++ Projects]/[Navigator] view.
Close Project Closes the project currently selected in the [C/C++ Projects]/[Navigator] view.
Build All (Ctrl+B) Executes a build process on all projects open in the [C/C++ Projects]/[Navigator] view.

Menu Bar

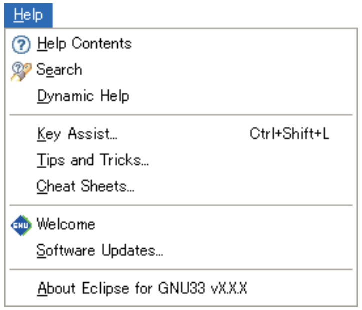
[Run] menu


Run
External Tools

[Window] menu


Window
New Window
New Editor
Open Perspective
Show View
Customize Perspective...
Save Perspective As...
Reset Perspective...
Close Perspective
Close All Perspectives
Navigation
Preferences

[Help] menu

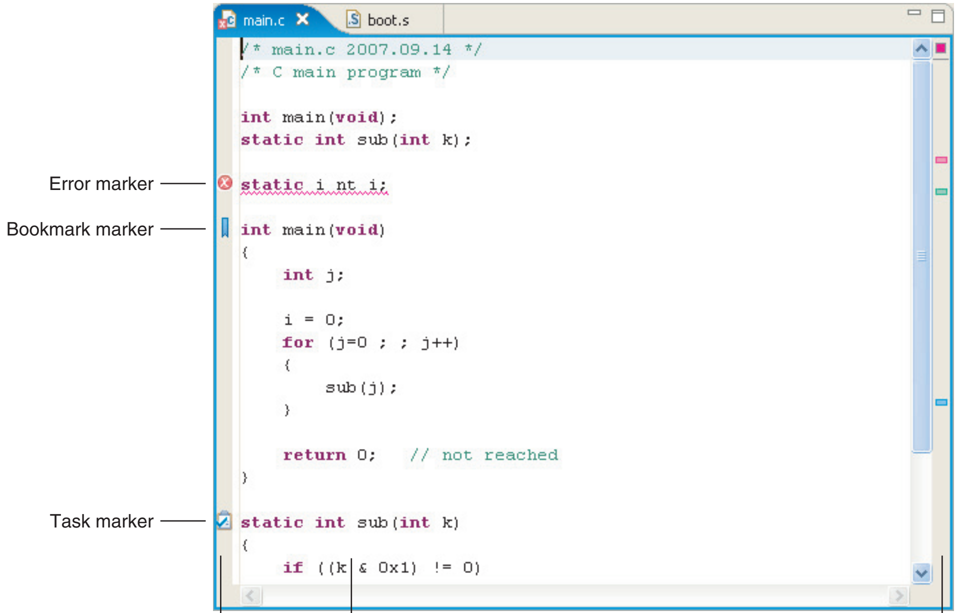

Help
Help Contents
Search
Dynamic Help
Key Assist... Ctrl+Shift+L
Tips and Tricks...
Cheat Sheets...
Welcome
Software Updates...
About Eclipse for GNU33 vX.X.X

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.
External Tools Starts the **gdb** debugger or an external tool.
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.
Preferences... Customizes the **IDE** environment.

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 GNU33 V3.*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/C++ editor or the assembler editor according to the file type to be edited.


main.c
boot.s
Error marker
Bookmark marker
Task marker

```
/* main.c 2007.09.14 */
/* C main program */

int main(void);
static int sub(int k);

static i nt i;

int main(void)
{
    int j;

    i = 0;
    for (j=0 ; ; j++)
    {
        sub(j);
    }

    return 0;   // not reached
}

static int sub(int k)
{
    if ((k & 0x1) != 0)
```

Marker bar

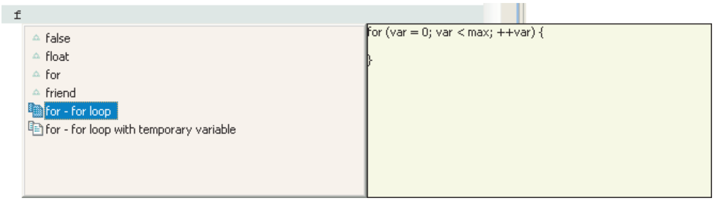
The marker bar shows the line in error and the markers indicating a bookmark, a line in which a task is set, etc. Markers are displayed on the left edge of the corresponding line.

Editing area

Edit sources in this area.
While a C/C++ source being edited, the content assist function shown below can be used by pressing the [Ctrl] + [Space] keys.

Overview ruler

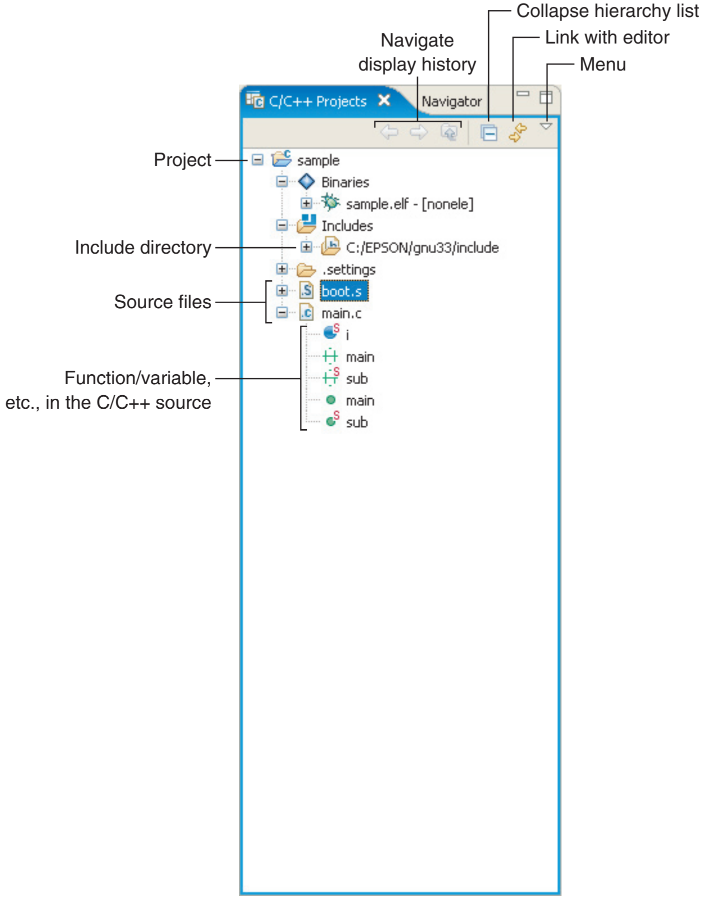
The overview ruler shows the position in error and the position at which a bookmark or task is set by a square symbol. Click a marker to go to that position.


f
false
float
for
friend
for - for loop
for - for loop with temporary variable

```
for (var = 0; var < max; ++var) {

}
```

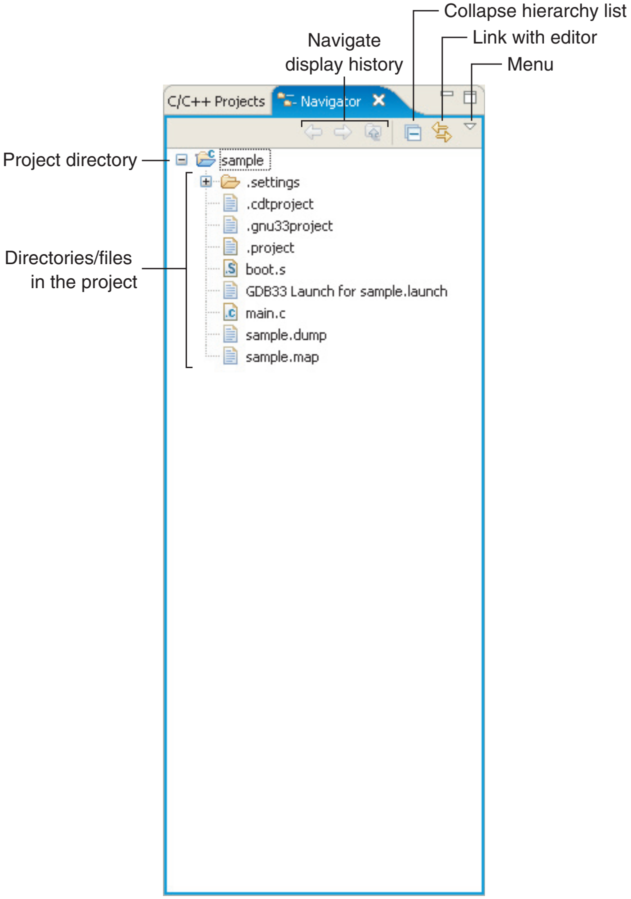
[C/C++ Projects] View

Lists the projects present in the workspace along with the C/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/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.


Collapse hierarchy list
Link with editor
Navigate display history
Menu
C/C++ Projects
Navigator
Project
sample
Binaries
sample.elf - [nonele]
Includes
Include directory
C:/EPSON/gnu33/include
.settings
Source files
boot.s
main.c
i
main
sub
main
sub
Function/variable, etc., in the C/C++ source

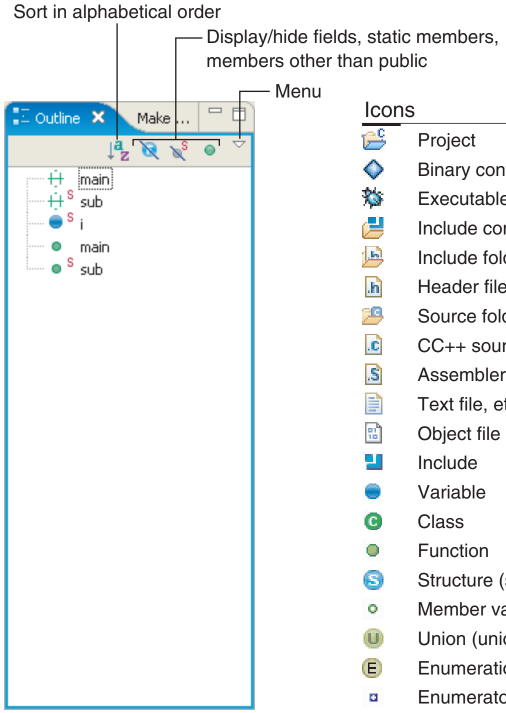
[Navigator] View

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.


Collapse hierarchy list
Link with editor
Navigate display history
Menu
C/C++ Projects
Navigator
Project directory
sample
.settings
.cdtproject
.gnu33project
.project
boot.s
GDB33 Launch for sample.launch
main.c
sample.dump
sample.map
Directories/files in the project

[Outline] View

Shows the functions, classes, and global variables that are written in the C/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.

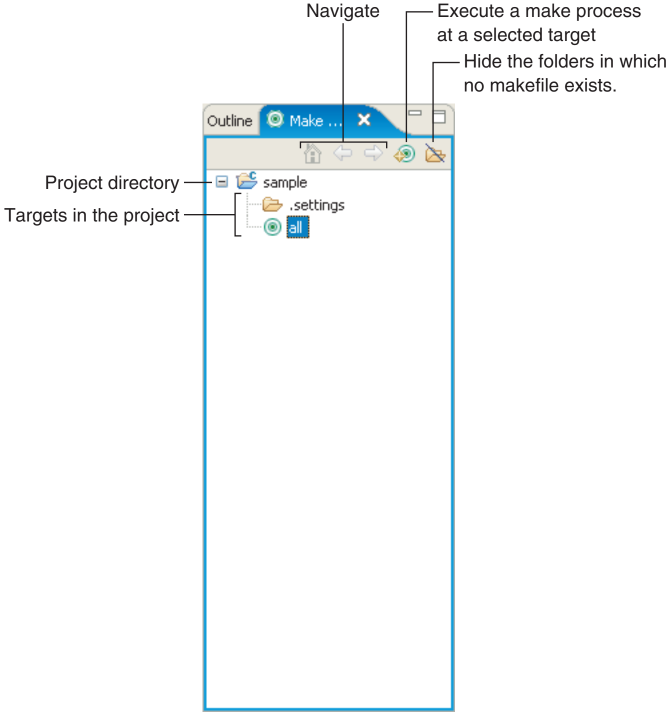

Sort in alphabetical order
Display/hide fields, static members, members other than public
Menu
Outline
Make ...
main
sub
i
main
sub

Icons

- Project
- Binary container
- Executable format file
- Include container
- Include folder
- Header file
- Source folder
- CC++ source file
- Assembler source file
- Text file, etc.
- Object file
- Include
- Variable
- Class
- Function
- Structure (struct)
- Member variable
- Union (union)
- Enumeration type (enum)
- Enumerator
- Function definition (prototype declaration)
- Macro-definition
- Type definition (typedef)
- Namespace

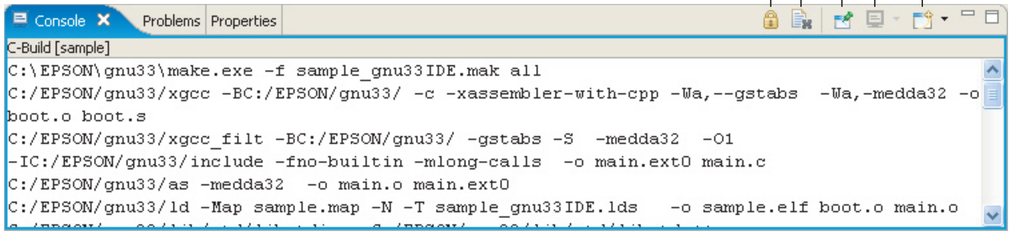
[Make Target] View

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
Outline
Make ...
sample
.settings
all

[Console] View

Displays the executed command line or the messages output by the GNU33 tools.



Clear the contents displayed

Automatic scroll lock

Enable other view while the [Console] view is displaying messages.

Switching [Console]

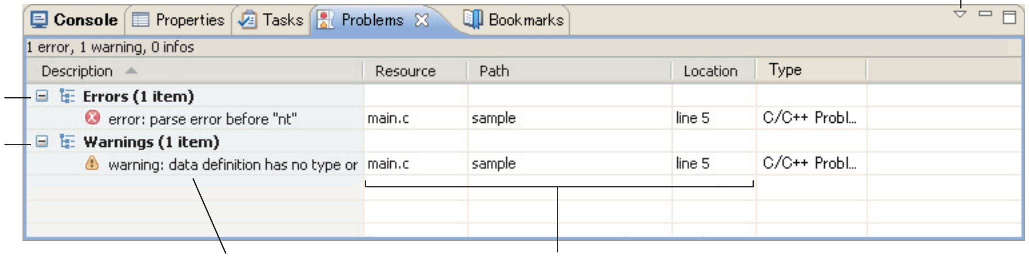
Opens a new [Console].

C-Build [sample]

```
C:\EPSON\gnu33\make.exe -f sample_gnu33IDE.mak all
C:/EPSON/gnu33/xgcc -BC:/EPSON/gnu33/ -c -xassembler-with-cpp -Wa,--gstabs  -Wa,-medda32 -o
boot.o boot.s
C:/EPSON/gnu33/xgcc_filt -BC:/EPSON/gnu33/ -gstabs -S  -medda32  -O1
-IC:/EPSON/gnu33/include -fno-builtin -mlong-calls  -o main.ext0 main.c
C:/EPSON/gnu33/as -medda32  -o main.o main.ext0
C:/EPSON/gnu33/ld -Map sample.map -N -T sample_gnu33IDE.lds   -o sample.elf boot.o main.o
```

[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.



Menu

1 error, 1 warning, 0 infos

Description	Resource	Path	Location	Type
⊟ **Errors (1 item)**				
error: parse error before "nt"	main.c	sample	line 5	C/C++ Probl...
⊟ **Warnings (1 item)**				
warning: data definition has no type or	main.c	sample	line 5	C/C++ Probl...

Error

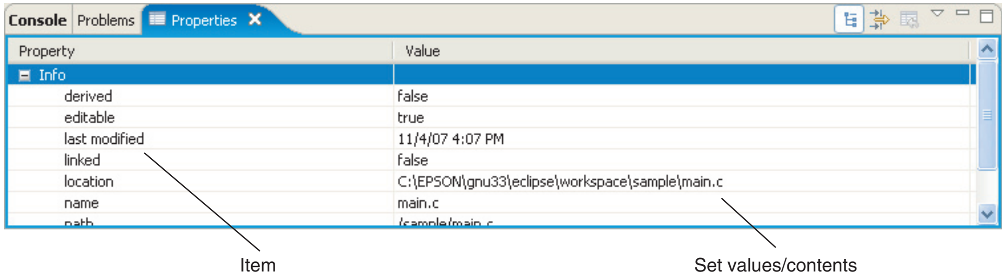
Warning

Error message

Locations in error (file/directory/line number)

[Properties] View

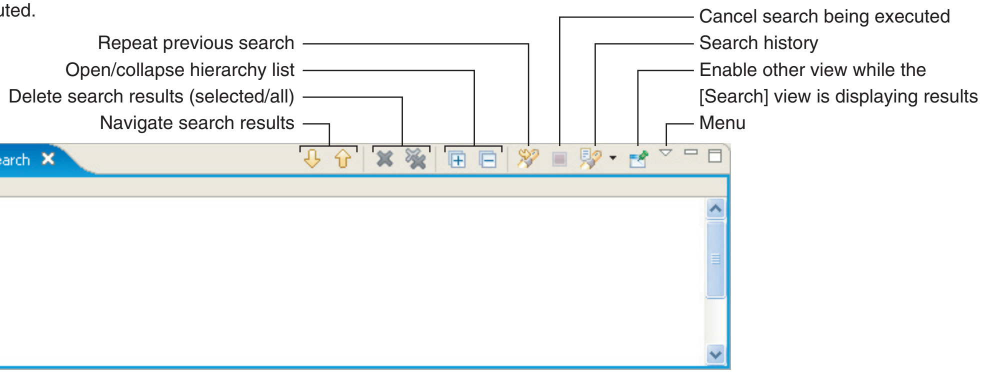
Displays information on the resource or member currently selected in the [C/C++ Projects], the [Navigator], or the [Outline] view.


Console
Problems
Properties
Item
Set values/contents

Property	Value
Info	
derived	false
editable	true
last modified	11/4/07 4:07 PM
linked	false
location	C:\EPSON\gnu33\eclipse\workspace\sample\main.c
name	main.c
path	/sample/main.c

[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.

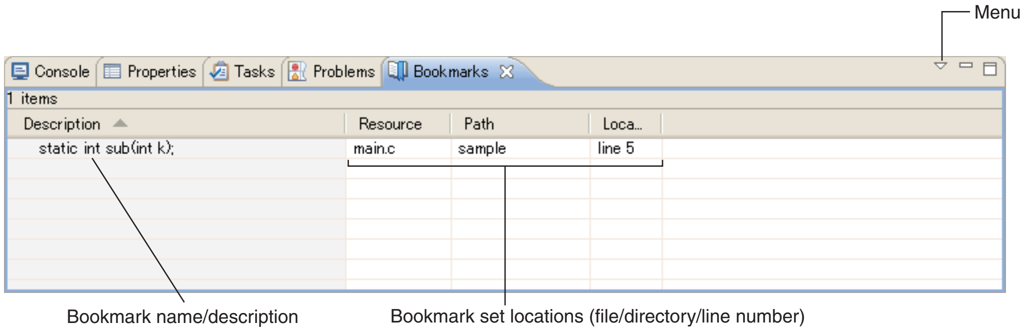

Repeat previous search
Open/collapse hierarchy list
Delete search results (selected/all)
Navigate search results
Cancel search being executed
Search history
Enable other view while the [Search] view is displaying results
Menu
Console
Problems
Properties
Search

'main' - 64 matches in workspace (*.*)

- sample
  - .gnu33project (8 matches)
  - boot.o
  - boot.s (2 matches)
  - main.c (4 matches)
  - main.d (2 matches)
  - main.ext0 (18 matches)

[Bookmarks] View

Shows the bookmarks registered in the editor, letting you jump to a bookmark or delete a bookmark.

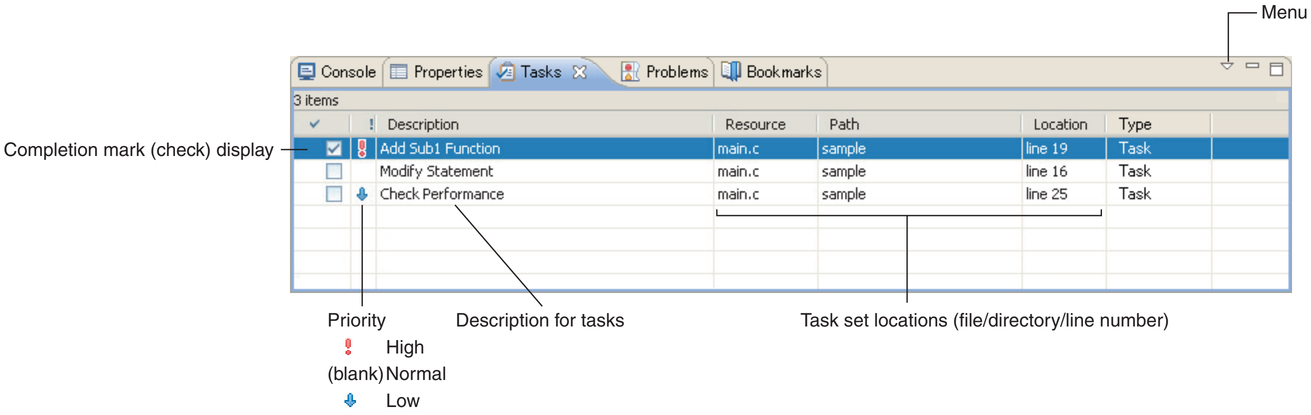

Menu
Console
Properties
Tasks
Problems
Bookmarks
Bookmark name/description
Bookmark set locations (file/directory/line number)

1 items

Description	Resource	Path	Loca...
static int sub(int k);	main.c	sample	line 5

[Tasks] View

Shows the tasks (To-Do) registered in the editor, letting you jump to or delete a task.


Menu
Console
Properties
Tasks
Problems
Bookmarks
Completion mark (check) display
Priority
❗ High
(blank) Normal
⬇ Low
Description for tasks
Task set locations (file/directory/line number)

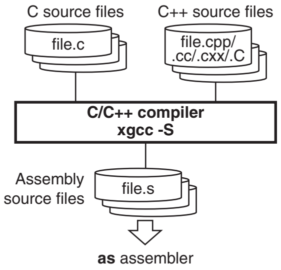
3 items

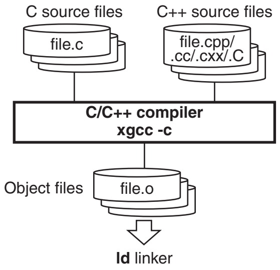
✓	!	Description	Resource	Path	Location	Type
☑	❗	Add Sub1 Function	main.c	sample	line 19	Task
☐		Modify Statement	main.c	sample	line 16	Task
☐	⬇	Check Performance	main.c	sample	line 25	Task

Outline

This tool is made based on GNU C/C++ Compiler and is compatible with ANSI C/C++. This tool invokes **cpp.exe** and **cc1.exe/cc1plus.exe** sequentially to compile C/C++ source files to the assembly source files for the S1C33 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


C source files
C++ source files
file.c
file.cpp/
.cc/.cxx/.C
C/C++ compiler
xgcc -S
Assembly
source files
file.s
as assembler


C source files
C++ source files
file.c
file.cpp/
.cc/.cxx/.C
C/C++ compiler
xgcc -c
Object files
file.o
ld linker

Start-up Command

`xgcc` *<options>* *<filename>*

<filename> C/C++ source file name

Examples: `xgcc -c -gstabs test.c` (C source)
`xgcc -c -gstabs test.cc` (C++ source)

Major Command-line Options

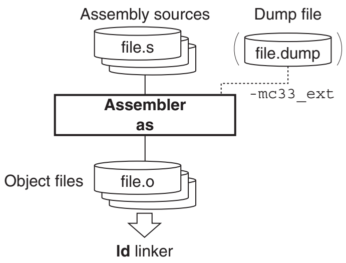
`-S`	Output assembly code (.s)
`-c`	Output relocatable object file (.o)
`-E`	Execute C preprocessor only
`-B`*<path>*	Specify compiler search path
`-I`*<path>*	Specify include file directory
`-fPIC`	Generate position-independent code
`-fno-builtin`	Disable built-in functions
`-D`*<macro>*[=*<string>*]	Define macro name
`-O0`, `-O`, `-O2`, `-O3`, `-Os`	Optimization
`-gstabs`	Add debugging information with relative path to source files
`-mc33adv`	Generate C33 ADV Core code
`-mc33401`	Assembler filter for the C33 ADV Core
`-mc33pe`	Generate C33 PE Core code
`-medda32`	Disable default data area
`-mlong-calls`	Extend function calls
`-mno-memcpy`	Inline expansion of strcpy and memcpy
`-Wall`	Enables warning options
`-mno-sjis-filt`	Disables filter function for Shift JIS code
`-xassembler-with-cpp`	Invoke C preprocessor

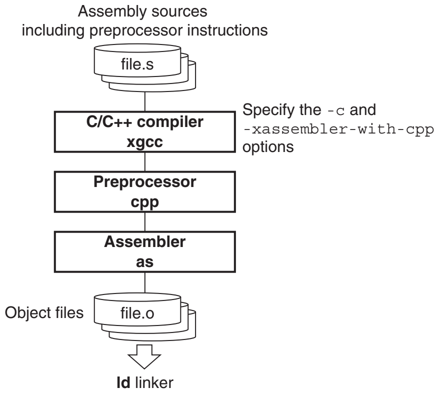
Assembler as (1)

Outline

This tool assembles assembly source files output by the C/C++ compiler and converts the mnemonics of the source files into object codes (machine language) of the S1C33000. 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 -mc33_ext test.dump allobjects.dump -o test.o test.s`

Major Command-line Options

`-o`*<filename>*	Specify output file name
`-a`[*<suboption>*]	Output assembly list file Example: `-adhl` (high-level assembly listing without debugging directives)
`--gstabs`	Add debugging information with relative path to source files
`-mc33adv`	Generate S1C33401 (C33 ADV Core) code
`-mc33401`	Assembler filter for the S1C33401 (C33 ADV Core)
`-mc33pe`	Generate C33 PE Core code
`-medda32`	Disable default data area
`-mc33_ext` *<dumpfile>* *<allobjfile>*	Optimize extended instructions

Major Preprocessor Pseudo-instructions

`#include`	Insertion of file
`#define`	Definition of character strings and numbers
`#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
`.section .comm`	Declare .comm section
`.section .ctors`	Declare .ctors section
`.section .dtors`	Declare .dtors section
`.section .gcc_except_table`	Declare .gcc_except_table 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

Symbol Masks

`@rh/@RH`	Acquires the 10 high-order bits of a relative address
`@rm/@RM`	Acquires the 13 mid-order bits of a relative address
`@rl/@RL`	Acquires the 8 low-order bits of a relative address
`@h/@H`	Acquires the 13 high-order bits of an absolute address
`@m/@M`	Acquires the 13 mid-order bits of an absolute address
`@l/@L`	Acquires the 6 low-order bits of an absolute address
`@ah/@AH`	Acquires the 13 high-order bits of a relative address
`@al/@AL`	Acquires the 13 low-order bits of a relative address

Pseudo-operands

`doff_hi()/DOFF_HI()`	Acquires the 13 high-order bits of an offset address from __dp
`doff_lo()/DOFF_LO()`	Acquires the 13 low-order bits of an offset address from __dp
`dpoff_h()/DPOFF_H()*`	Acquires the 13 high-order bits of an offset address from __dp
`dpoff_m()/DPOFF_M()*`	Acquires the 13 mid-order bits of an offset address from __dp
`dpoff_l()/DPOFF_L()*`	Acquires the 6 low-order bits of an offset address from __dp

(* for the S1C33401 (C33 ADV Core))

Error/Warning Messages

Error messages

Unrecognized opcode: '*XXXXX*'	The operation code *XXXXX* is undefined.
XXXXXX: invalid operand	A format error of the operand.
XXXXXX: invalid register name	The specified register cannot be used.
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 `malloc()` 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 `-mc33_ext` option is specified. Or the specified dump file does not exist.
The format of the dump file is invalid.	The contents in the dump file specified with the `-mc33_ext` option are invalid.
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 `-mc33_ext` option has exceeded 2,047 characters.
Cannot find the all objects' dump file.	An all-object dump file name is not specified even though the `-mc33_ext` option is specified. Or the specified file does not exist.
The format of the all objects' dump file is invalid.	The contents in the all-object dump file specified with the `-mc33_ext` option are invalid.

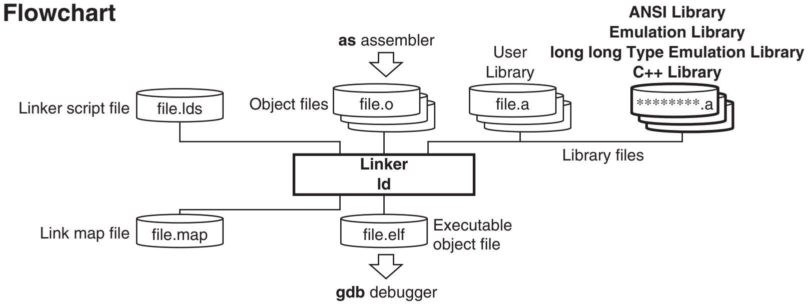
Warning messages

operand out of range (*XXXXXX*: *XXX* not between *AAA* and *BBB*)	The value specified in the operand is out of the effective range. It has been corrected within the range (*AAA*–*BBB*).
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. `.long`, `.int`) 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*.

Outline

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

Flowchart


as assembler
ANSI Library
Emulation Library
long long Type Emulation Library
C++ Library
User Library
Linker script file
file.lds
Object files
file.o
file.a
********.a
Library files
Linker
ld
Link map file
file.map
file.elf
Executable object file
gdb debugger

Start-up Command

`ld` *<options> <filename>*

<filename> Object and library files to be linked

Example:
```
ld -o sample.elf boot.o sample.o ..\lib\libc.a
..\lib\libgcc.a -defsym _dp=0
```

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/Warning Messages

Error messages

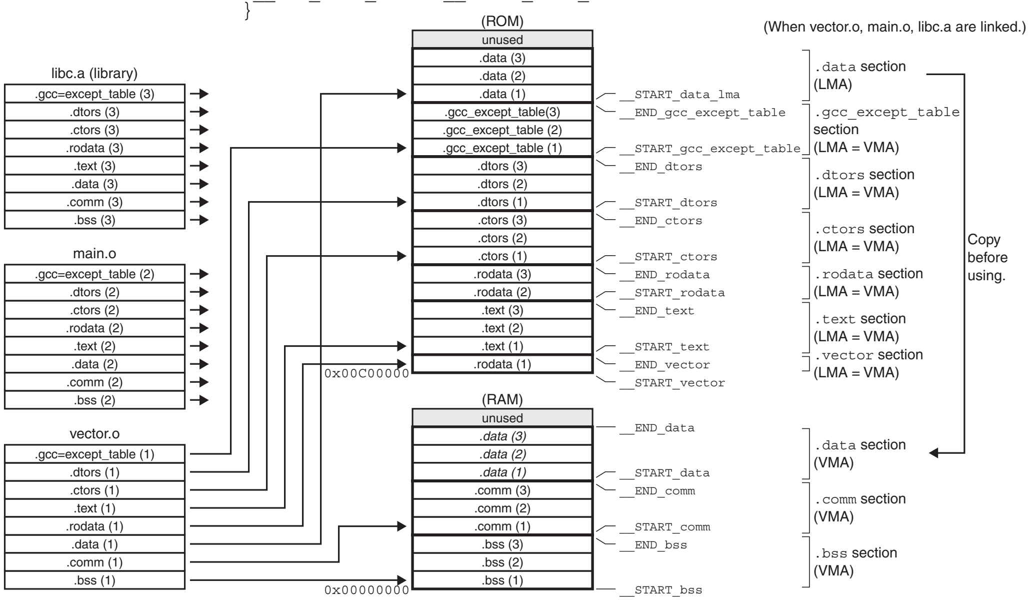
Default Data area pointer value is larger than symbol address value.	The `__dp` (default data area pointer) value exceeds the defined symbol address.
The offset value of a symbol is over 64MB. (default data area)	The symbol-offset value is out of the 64MB range from `__dp` (default data area pointer).
The offset value of a symbol is over 512KB. (default data area)	The symbol-offset value is out of the 512KB range from `__dp` (default data area pointer).
The offset value of a symbol is over 64byte. (default data area)	The symbol-offset value is out of the 64-byte range from `__dp` (default data area pointer).
Cannot link [STD|PE|ADV] object *<objectfile>* included from *<archivefile>* with [STD|PE|ADV] object *<first objectfile>*	The object file *<objectfile>* included in the archived file *<archivefile>* cannot be linked with *<first objectfile>*, as the target CPU is different.
Input object file *<objectfile>* [included from *<archivefile>*] is not for C33.	The object file *<objectfile>* included in the archived file *<archivefile>* is not a C33 object file.

Warning message

__dp symbol cannot be referred to.	1. The `doff_hi`, `dpoff_h`, `dpoff_m`, `dpoff_l` or `doff_lo` pseudo-operand is used without specifying `__dp`. 2. A `[`*`symbol+imm`*`]` operand is included in the assembler source without specifying `__dp`. `__dp` will be treated as 0x0.

Default linker script file generated by the IDE

```
OUTPUT_FORMAT("elf32-c33", "elf32-c33", "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);

SECTIONS
{
 /* data pointer symbols */
 __dp = 0x00000000;

 /* location counter */
 . = 0x0;

 /* section information */
 .bss 0x00000000 :
   {
    __START_bss = . ;
    boot.o(.bss)   )
    main.o(.bss)   ) files(.bss)
    libc.a(.bss)   )
   }
   __END_bss = . ;

 .comm __END_bss :
   {
    __START_comm = . ;
    files(.comm)
   }
   __END_comm = . ;

 .data __END_comm : AT( __END_gcc_except_table )
   {
    __START_data = . ;
    files(.data)
   }
   __END_data = . ;

 .vector 0x00C00000 :
   {
    __START_vector = . ;
    boot.o(.rodata)
   }
   __END_vector = . ;

 .text __END_vector :
   {
    __START_text = . ;
    files(.text)
   }
   __END_text = . ;

 .rodata __END_text :
   {
    __START_rodata = . ;
    main.o(.rodata)
    libc.a(.rodata)
   }
   __END_rodata = . ;

 .ctors __END_rodata :
   {
    . = ALIGN(4);
    __START_ctors = . ;
    files(.ctors)
   }
   __END_ctors = . ;

 .dtors __END_ctors :
   {
    . = ALIGN(4);
    __START_dtors = . ;
    files(.dtors)
   }
   __END_dtors = . ;

 .gcc_except_table __END_dtors :
   {
    __START_gcc_except_table = . ;
    files(.gcc_except_table)
   }
   __END_gcc_except_table = . ;

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

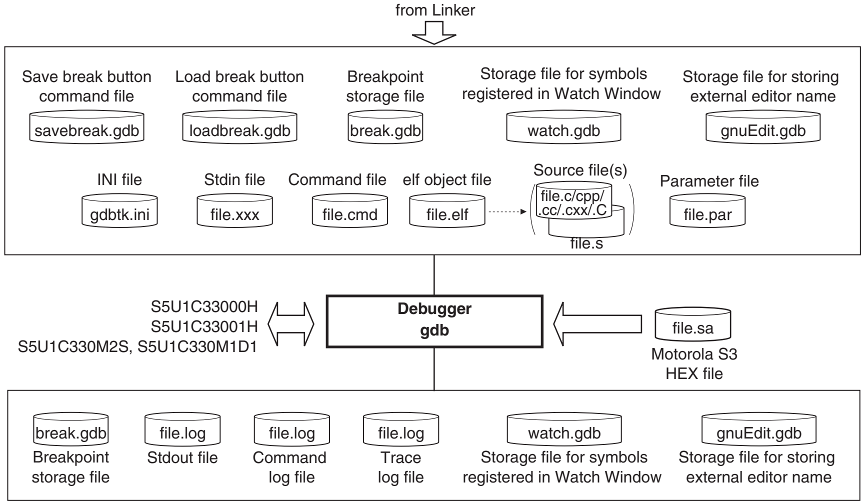

(When vector.o, main.o, libc.a are linked.)
libc.a (library)
.gcc=except_table (3)
.dtors (3)
.ctors (3)
.rodata (3)
.text (3)
.data (3)
.comm (3)
.bss (3)
main.o
.gcc=except_table (2)
.dtors (2)
.ctors (2)
.rodata (2)
.text (2)
.data (2)
.comm (2)
.bss (2)
vector.o
.gcc=except_table (1)
.dtors (1)
.ctors (1)
.text (1)
.rodata (1)
.data (1)
.comm (1)
.bss (1)
(ROM)
unused
.data (3)
.data (2)
.data (1)
.gcc_except_table(3)
.gcc_except_table (2)
.gcc_except_table (1)
.dtors (3)
.dtors (2)
.dtors (1)
.ctors (3)
.ctors (2)
.ctors (1)
.rodata (3)
.rodata (2)
.text (3)
.text (2)
.text (1)
.rodata (1)
0x00C00000
(RAM)
unused
.data (3)
.data (2)
.data (1)
.comm (3)
.comm (2)
.comm (1)
.bss (3)
.bss (2)
.bss (1)
0x00000000
__START_data_lma
__END_gcc_except_table
__START_gcc_except_table
__END_dtors
__START_dtors
__END_ctors
__START_ctors
__END_rodata
__START_rodata
__END_text
__START_text
__END_vector
__START_vector
__END_data
__START_data
__END_comm
__START_comm
__END_bss
__START_bss
.data section (LMA)
.gcc_except_table section (LMA = VMA)
.dtors section (LMA = VMA)
.ctors section (LMA = VMA)
.rodata section (LMA = VMA)
.text section (LMA = VMA)
.vector section (LMA = VMA)
.data section (VMA)
.comm section (VMA)
.bss section (VMA)
Copy before using.

Debugger gdb (1)

Outline

The **gdb** serves to perform source-level debugging by controlling the hardware tool (S5U1C33001H, S5U1C33000H) or debug monitor (S5U1C330M2S). 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


from Linker
Save break button command file
savebreak.gdb
Load break button command file
loadbreak.gdb
Breakpoint storage file
break.gdb
Storage file for symbols registered in Watch Window
watch.gdb
Storage file for storing external editor name
gnuEdit.gdb
INI file
gdbtk.ini
Stdin file
file.xxx
Command file
file.cmd
elf object file
file.elf
Source file(s)
file.c/cpp/.cc/.cxx/.C
file.s
Parameter file
file.par
S5U1C33000H
S5U1C33001H
S5U1C330M2S, S5U1C330M1D1
Debugger gdb
file.sa
Motorola S3 HEX file
break.gdb
Breakpoint storage file
file.log
Stdout file
file.log
Command log file
file.log
Trace log file
watch.gdb
Storage file for symbols registered in Watch Window
gnuEdit.gdb
Storage file for storing external editor name

Start-up Command

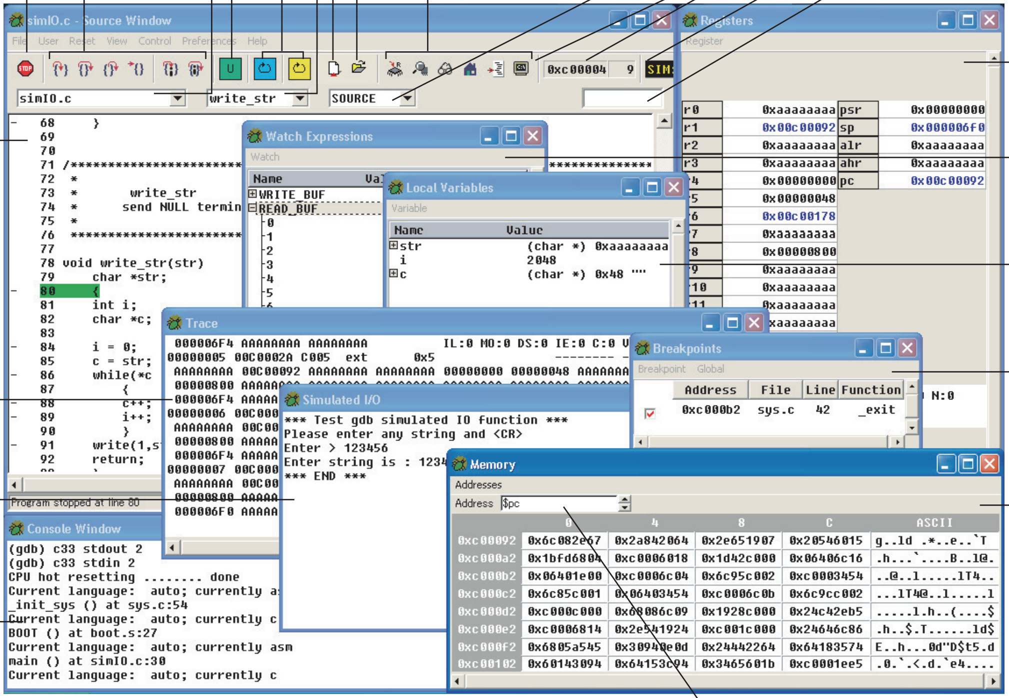
`gdb` *<options>*

Example: `gdb -x sample.cmd --cd=/cygdrive/c/EPSON/gnu33/sample`

Command-line Options

`--command=`*<filename>*	Specifies a command file
`-x` *<filename>*	Specifies a command file
`--c33_cmw=`*<seconds>*	Specifies the command execution intervals for command files
`--cd=`*<path>*	Changes current directory
`--directory=`*<path>*	Changes source file directory
`--c33_euc`	Displays the EUC code
`--double_starting`	Enables double starting

Windows



Reset buttons

User defined button

Combo box to display and search source file names

Forced break button

Program execution buttons

Combo box to display and search function names

[Save Break / Watch] button

[Load break / Watch] button

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

[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.

Text box for specifying display start address

simIO.c - Source Window

```
68      }
69
70
71 /************************
72  *
73  *       write_str
74  *      send NULL termin
75  *
76  ************************
77
78 void write_str(str)
79      char *str;
80      {
81      int i;
82      char *c;
83
84      i = 0;
85      c = str;
86      while(*c
87          {
88          c++;
89          i++;
90          }
91      write(1,s
92      return;
```

Program stopped at line 80

Watch Expressions

```
Name                Va
⊞WRITE_BUF
⊟READ_BUF
  ├0
  ├1
  ├2
  ├3
  ├4
  ├5
  ├6
```

Local Variables

```
Name           Value
⊞str           (char *) 0xaaaaaaaa
 i             2048
⊞c             (char *) 0x48 ""
```

Registers

r0	0xaaaaaaaa	psr	0x00000000
r1	0x00c00092	sp	0x000006f0
r2	0xaaaaaaaa	alr	0xaaaaaaaa
r3	0xaaaaaaaa	ahr	0xaaaaaaaa
4	0x00000000	pc	0x00c00092
5	0x00000048		
6	0x00c00178		
7	0xaaaaaaaa		
8	0x00000800		
9	0xaaaaaaaa		
10	0xaaaaaaaa		
11	0xaaaaaaaa		
	xaaaaaaaa		

N:0

Trace

```
 000006F4 AAAAAAAA AAAAAAAA                  IL:0 MO:0 DS:0 IE:0 C:0 V
00000005 00C0002A C005  ext       0x5                        ---------
 AAAAAAAA 00C00092 AAAAAAAA AAAAAAAA 00000000 00000048 AAAAAAA
 00000800 AAAAA
 000006F4 AAAAA
00000006 00C000
 AAAAAAAA 00C00
 00000800 AAAAA
 000006F4 AAAAA
00000007 00C000
 AAAAAAAA 00C00
 00000800 AAAAA
 000006F0 AAAAA
```

Simulated I/O

```
*** Test gdb simulated IO function ***
Please enter any string and <CR>
Enter > 123456
Enter string is : 1234
*** END ***
```

Breakpoints

	Address	File	Line	Function
☑	0xc000b2	sys.c	42	_exit

Console Window

```
(gdb) c33 stdout 2
(gdb) c33 stdin 2
CPU hot resetting ........ done
Current language:  auto; currently a
_init_sys () at sys.c:54
Current language:  auto; currently c
BOOT () at boot.s:27
Current language:  auto; currently asm
main () at simIO.c:30
Current language:  auto; currently c
```

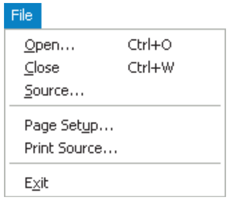
Memory

Address $pc

	0	4	8	C	ASCII
0xc00092	0x6c082e67	0x2a842064	0x2e651907	0x20546015	g..ld .*..e..`T
0xc000a2	0x1bfd6804	0xc0006018	0x1d42c000	0x06406c16	.h...`....B..l@.
0xc000b2	0x06401e00	0xc0006c04	0x6c95c002	0xc0003454	..@..l.....lT4..
0xc000c2	0x6c85c001	0x06403454	0xc0006c0b	0x6c9cc002	...lT4@..l.....l
0xc000d2	0xc000c000	0x68086c09	0x1928c000	0x24c42eb5	.....l.h..(....$
0xc000e2	0xc0006814	0x2e541924	0xc001c000	0x24646c86	.h..$.T......ld$
0xc000f2	0x6805a545	0x30940e0d	0x24442264	0x64183574	E..h...0d"D$t5.d
0xc00102	0x60143094	0x64153c94	0x3465601b	0xc0001ee5	.0.`.<.d.`e4....

Menus

[Source] window


File
Open... Ctrl+O
Close Ctrl+W
Source...
Page Setup...
Print Source...
Exit

[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
User Command

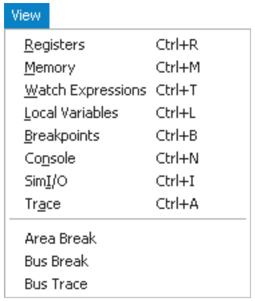
[User] menu

User Command Executes the command file userdefine.gdb.

Reset
Reset cold
Reset hot

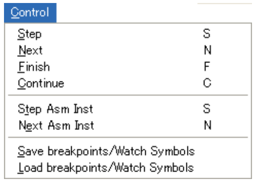
[Reset] menu

Reset cold Executes the command file resetcold.gdb (`c33 rstc` command).
Reset hot Executes the command file resethot.gdb (`c33 rsth` command).


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
Area Break
Bus Break
Bus Trace

[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.
SimI/O Opens the [Simulated I/O] window.
Trace Opens the [Trace] window.
Area Break Opens the [On Chip Area Break] dialog box used to set the on-chip area break.
Bus Break Opens the [On Chip Bus Break] dialog box used to set the on-chip bus break.
Bus Trace Opens the [On Chip Bus Trace] dialog box used to set the on-chip bus trace.

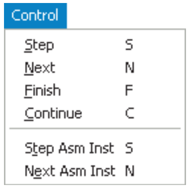

Control
Step S
Next N
Finish F
Continue C
Step Asm Inst S
Next Asm Inst N
Save breakpoints/Watch Symbols
Load breakpoints/Watch Symbols

[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.
Save breakpoints / Watch Symbols The gdb debugger stores the breakpoint status and symbol names registered in the Watch Window.
Load breakpoints / Watch Symbols Sets the breakpoints and the symbol names in the Watch Window as stored in the gdb debugger.

Menus

[Source] window


Control
Step S
Next N
Finish F
Continue C
Step Asm Inst S
Next Asm Inst N

[Control] menu (continued)

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.

Preferences
Global...
Source...
Editor Selection...

[Preferences] menu

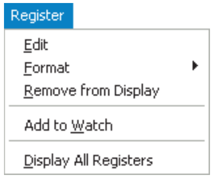
Global... Displays the [Global Preferences] dialog box used to select the fonts.
Source... Displays the [Source Preferences] dialog box for selecting source display colors and tab width.
Editor Selection Displays the [Editor Selection] dialog box used for setting the external editor to use.

Help
About GDB...

[Help] menu

About GDB... Displays the version of the debugger.

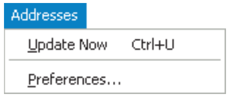
[Registers] window


Register
Edit
Format
Remove from Display
Add to Watch
Display All Registers

[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.

[Memory] window

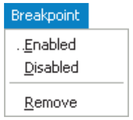

Addresses
Update Now Ctrl+U
Preferences...

[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.

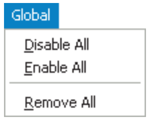
Menus

[Breakpoints] window



[Breakpoint] menu

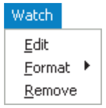
Enabled Enables the breakpoint selected in the window.
Disabled Disables the breakpoint selected in the window.
Remove Clears the breakpoint selected in the window.



[Global] menu

Disable All Disables all the software PC breakpoints that have been set.
Enable All Enables all the software PC breakpoints that have been set.
Remove All Clears all the software PC breakpoints that have been set.

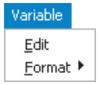
[Watch Expressions] window



[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 `step` command in Source or SRC+ASM display mode (otherwise, `stepi`).
`N`	Executes the `next` command in Source or SRC+ASM display mode (otherwise, `nexti`).
`F`	Execute the `finish` command.
`C`	Execute the `continue` 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

c33 fb *addr1 addr2 data*	Fills a memory area. (byte)	ICD2/3/6/SIM/MON
c33 fh *addr1 addr2 data*	Fills a memory area. (half word)	ICD2/3/6/SIM/MON
c33 fw *addr1 addr2 data*	Fills a memory area. (word)	ICD2/3/6/SIM/MON
x /[*length*]**b** [*addr*]	Dumps memory data. (byte)	ICD2/3/6/SIM/MON
x /[*length*]**h** [*addr*]	Dumps memory data. (half word)	ICD2/3/6/SIM/MON
x /[*length*]**w** [*addr*]	Dumps memory data. (word)	ICD2/3/6/SIM/MON
set {char} *addr*=*data*	Sets memory data. (byte)	ICD2/3/6/SIM/MON
set {short} *addr*=*data*	Sets memory data. (half word)	ICD2/3/6/SIM/MON
set {int} *addr*=*data*	Sets memory data. (word)	ICD2/3/6/SIM/MON
c33 mvb *addr1 addr2 addr3*	Copies memory area. (byte)	ICD2/3/6/SIM/MON
c33 mvh *addr1 addr2 addr3*	Copies memory area. (half word)	ICD2/3/6/SIM/MON
c33 mvw *addr1 addr2 addr3*	Copies memory area. (word)	ICD2/3/6/SIM/MON
c33 df *addr1 addr2 type file* [*ap*]	Saves memory data to file.	ICD2/3/6/SIM/MON
c33 rm *addr1 addr2*	Reads target memory.	ICD2/3/6
c33 readmd *mode*	Memory read mode	ICD3/6

Register operation

info reg [*register*]	Displays register data.	ICD2/3/6/SIM/MON
set $_register_=_data_	Sets register data.	ICD2/3/6/SIM/MON

Program execution

continue [*Ignore*]	Executes program successively.	ICD2/3/6/SIM/MON
until *addr*	Executes program successively with temporary break.	ICD2/3/6/SIM/MON
step [*count*]	Executes source lines.	ICD2/3/6/SIM/MON
stepi [*count*]	Executes instruction steps.	ICD2/3/6/SIM/MON
next [*count*]	Executes source lines with function skip.	ICD2/3/6/SIM/MON
nexti [*count*]	Executes instruction steps with subroutine skip.	ICD2/3/6/SIM/MON
finish	Exits from function/subroutine.	ICD2/3/6/SIM/MON
c33 callmd *mode* [*file*]	Sets user-function call mode.	ICD2/3/6/SIM/MON
c33 call *func* [*arg1*... [*arg3*]]]	Calls user function.	ICD2/3/6/SIM/MON

CPU reset

c33 rstc	Cold-resets CPU (executes resetcold.gdb).	ICD2/3/6/SIM/MON
c33 rsth	Hot-resets CPU (executes resethot.gdb).	ICD2/3/6/SIM/MON
c33 rstt	Reset target.	ICD6

Interrupt

c33 int [*type level*]	Generates interrupt.	SIM

Break

break [*addr*]	Sets software PC breakpoint.	ICD2/3/6/SIM/MON
tbreak [*addr*]	Sets temporary software PC breakpoint.	ICD2/3/6/SIM/MON
hbreak [*addr*]	Sets hardware PC breakpoint.	ICD2/3/6/SIM/MON
thbreak [*addr*]	Sets temporary hardware PC breakpoint.	ICD2/3/6/SIM/MON
watch *addr*	Sets data-write breakpoint.	ICD2/3/6/SIM/MON
rwatch *addr*	Sets data-read breakpoint.	ICD2/3/6/SIM/MON
awatch *addr*	Sets data-read/write breakpoint.	ICD2/3/6/SIM/MON
delete [*breakNo.*]	Clears breakpoint by break number.	ICD2/3/6/SIM/MON
clear [*addr*]	Clears breakpoint by location.	ICD2/3/6/SIM/MON
enable [*breakNo.*]	Enables breakpoint.	ICD2/3/6/SIM/MON
disable [*breakNo.*]	Disables breakpoint.	ICD2/3/6/SIM/MON
ignore *breakNo. count*	Disables breakpoint with ignore count.	ICD2/3/6/SIM/MON
info breakpoints	Displays breakpoint list.	ICD2/3/6/SIM/MON
c33 oab *parameters...*	Sets on chip area (CE) break. *1	ICD3/6
c33 obb *parameters...*	Sets on chip bus break. *1	ICD3/6
c33 timebrk *timer*	Set lapse of time break	ICD6
c33 savebreak	Save set breakpoints	ICD2/3/6/SIM/MON
c33 loadbreak	Restore saved breakpoints	ICD2/3/6/SIM/MON

Symbol information

info locals	Displays local symbol information.	ICD2/3/6/SIM/MON
info var	Displays global symbol information.	ICD2/3/6/SIM/MON
print *symbol*[=*value*]	Changes symbol values.	ICD2/3/6/SIM/MON

File

file *file*	Loads debug information.	ICD2/3/6/SIM/MON
load [*file*]	Loads program.	ICD2/3/6/SIM/MON

Map information

c33 rpf *file*	Sets map information.	ICD2/3/6/SIM/MON
c33 map	Displays map information.	ICD2/3/6/SIM/MON

Flash memory

c33 fls *addr1 addr2 erase write*	Sets up flash memory.	ICD2/3/6/MON
c33 fle *control block1 block2* [*timer*]	Erases flash memory.	ICD2/3/6/MON

Trace

c33 tm *md1 md2* [*trg1 trg2 md3 md4*]	Sets trace mode. (ICD mode)	ICD2/3/6
c33 tm on/off [*mode* [*file*]]	Sets trace mode. (simulator mode)	SIM
c33 td [*cycle1 cycle2*]	Displays trace information.	ICD2/3/6

Debug Commands

Trace

c33 ts *addr* [*pre post*]	Searches trace information.	ICD2/3/6
c33 tf *file* [*cycle1 cycle2*]	Saves trace information.	ICD2/3/6
c33 obt *parameters…*	Sets on-chip bus trace mode. *1	ICD3/6
c33 otd *mode* [*cycle1 cycle2*]	Displays on-chip bus trace contents. *1	ICD3/6
c33 otf *mode file* [*cycle1 cycle2*]	Saves on-chip bus trace contents. *1	ICD3/6

Simulated I/O

c33 stdin 1/**2** *break buffer* [*file*]	Sets the simulated input condition.	ICD2/3/6/SIM/MON
c33 stdout 1/**2** *break buffer* [*file*]	Sets the simulated output condition.	ICD2/3/6/SIM/MON

Flash writer

c33 fwe 0/**1**	Erases program/data.	ICD3/6
c33 fwlp *file erase write* [*comment*]	Loads program.	ICD3/6
c33 fwld *file blk1 blk2 par* [*comment*]	Loads data.	ICD3/6
c33 fwdc *addr sz blk1 blk2 par* [*comm*]	Copies target memory.	ICD3/6
c33 fwd	Displays flash writer information.	ICD3/6

Others

c33 log [*file*]	Logging	ICD2/3/6/SIM/MON
source *file*	Executes command file.	ICD2/3/6/SIM/MON
c33 clockmd *mode func*	Sets execution counter mode.	ICD2/3/6/SIM
c33 clock	Displays execution counter.	ICD2/3/6/SIM
target *type* [*time*]	Connects target.	ICD2/3/6/SIM/MON
detach	Disconnects target.	ICD2/3/6/SIM/MON
c33 das *ROMaddr RAMaddr* [*CPU*]	Sets debug unit address.	ICD3/6
c33 lpt [*port*]	Sets/clears parallel loading mode.	ICD2
pwd	Displays current directory.	ICD2/3/6/SIM/MON
cd *directory*	Changes current directory.	ICD2/3/6/SIM/MON
c33 firmupdate *file*	Updates ICD firmware.	ICD3/6
c33 dclk *number*	Changes DCLK cycle.	ICD2/3/6/MON
c33 oscwait [*counter*]	Sets OSC1 wait counter.	ICD3/6
c33 cachehit	Displays cache hit rate. *1	ICD3/6
c33 logiana *mode user*	Sets logic analyzer mode.	ICD3/6
c33 help [*Command/GroupNo.*]	Help	ICD2/3/6/SIM/MON
c33 savewatch	Store symbols registered in Watch Window	ICD2/3/6/SIM/MON
c33 loadwatch	Register stored symbols to Watch Window	ICD2/3/6/SIM/MON
quit	Terminates debugger.	ICD2/3/6/SIM/MON

*1 Usable only when debugging the S1C33401 (C33 ADV Core).

Status and Error Messages

Status messages

Breakpoint *#*, *function* at *file*:*line*	Made to break at the set breakpoint.
Break by hard pc break1.	Made to break at hardware PC breakpoint 1.
Break by hard pc break2.	Made to break at hardware PC breakpoint 2.
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.
Break by sleep or halt.	Made to break by the execution of an `slp` or `halt` instruction in simulator mode.
Illegal address exception.	Made to break by the execution of an illegal address instruction 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.
Hardware read watchpoint *#*: *symbol*	Made to break by a data read condition.
Hardware access (read/write) watchpoint *#*: *symbol*	Made to break by a data access (read/write) condition.
Program received signal SIGTRAP, Trace/breakpoint trap.	Forcibly made to break using the [Stop] button. (Simulator mode)
Program received signal SIGINT, Interrupt.	Forcibly made to break using the [Stop] button. (ICD mode)

Error messages

A setup of a serial port was not completed.	The serial communication rate that is set for communication with the ICD is not 115,200 or 38,400 bps.
Address(0x*#*) is ext or delayed instruction.	The specified address cannot be set, as it is for the `ext` or delayed instruction.
C33 command error, command is not supported in present mode.	The input command cannot be executed in the currently supported connection mode.
C33 command error, command is too long.	The input command is too long, as it exceeds 256 characters.
C33 command error, invalid command.	The command is invalid; it contains an error.
C33 command error, number of parameter.	The number of parameters in the command is incorrect.
C33 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.
C33 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 #	The address # cannot be accessed.
Cannot allocate memory.	A memory area of the size specified by a parameter cannot be allocated.
Cannot clear data break(0x*#*).	The specified data break address is nonexistent.
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.
Cannot open file(*file*).	The system cannot open the file.
Cannot open ICD33 usb driver.	The system failed to open the USB driver.
Cannot set data break any more.	The system cannot set any more data breakpoints.
Cannot set hard pc break any more.	The number of hardware PC breakpoints set exceeds the limit (up to 2).
Cannot set soft pc break any more.	The number of software PC breakpoints set exceeds the limit (up to 200).
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 error(host->ICD33).	The system failed in USB transmission to the ICD.
Communication internal error(*#*).	An internal error occurred during communication with the ICD.
Communication system error(*#*).	The connection was unexpectedly terminated during communication with the ICD.

Status and Error Messages

Error messages

Framing error.	A framing error occurred during communication with the ICD.
ICD33 is busy(*#*).	The ICD side is busy.
Initialization error of ICD33.	The system failed to initialize the target.
Invalid parameter file(*#*: *file*).	The parameter file contains an error.
Invalid parameter file, start address > end address(*#*: *file*).	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.
It has not connected with a target.	The ICD cannot be connected to the target.
It is not c33 architecture ELF file.	The file specified with the `file` command is not an elf format file supported in S5U1C33001C.
Load file is not motorola format.(*file*)	The file is not a Motorola-format file.
Load max address(0x*#*) overflow.	The addresses to be loaded exceed the maximum value.
Load motorola file format error.(*file*)	The Motorola file to be loaded contains a format error.
Max address(0x*#*) overflow.	The specified addresses exceed the maximum value.
No memory map information.	No memory-map information is available.
Receiving message is inaccurate.	A message exceeding the maximum size was received during communication with the ICD.
Size limit(0x*#*) over.	The specified size exceeds the maximum value.
Size limit(0x*#*-0x*#*) over.	The specified size is outside the valid range.
Specification is required in the device for connecting.	There is an insufficient number of device names from which to select the ICD in the `target` command.
Target connection causes communication error at *#* times.	An error was encountered during an SIO connection check.
Target connection causes data error at *#* times.	The system cannot execute an SIO connection check.
Target connection causes time out error at *#* times.	The SIO connection check has timed out.
There is no argument given to this command.	The system failed to disconnect the target.
Timeout error #(ICD33 -> host).	In communication with the ICD, a receive operation timed out during the wait for data from the ICD.
Transmitting failure(*#*).	NAK was received from the ICD during communication with the ICD.
Expression cannot be implemented with read/access watchpoint.	The data break cannot be set for the specified address.
Please input "Editor path".	Enter characters in "Enter path".
"Editor path" exceeds 255 characters.	The number of characters entered in "Editor path" exceeds 255.
gnuEdit.gdb file save error	The external editor path could not be saved.

Floating-point Emulation Library libgcc.a

Double-type operation

__adddf3	Addition	(%r5, %r4) ← (%r7, %r6) + (%r9, %r8)
__subdf3	Subtraction	(%r5, %r4) ← (%r7, %r6) - (%r9, %r8)
__muldf3	Multiplication	(%r5, %r4) ← (%r7, %r6) * (%r9, %r8)
__divdf3	Division	(%r5, %r4) ← (%r7, %r6) / (%r9, %r8)
__negdf2	Sign change	(%r5, %r4) ← -(%r7, %r6)

Float-type operation

__addsf3	Addition	%r4 ← %r6 + %r7
__subsf3	Subtraction	%r4 ← %r6 - %r7
__mulsf3	Multiplication	%r4 ← %r6 * %r7
__divsf3	Division	%r4 ← %r6 / %r7
__negsf2	Sign change	%r4 ← -%r6

Type conversion

__fixunsdfsi	double → unsigned int	%r4 ← (%r7, %r6)
__fixdfsi	double → int	%r4 ← (%r7, %r6)
__floatsidf	int → double	(%r5, %r4) ← %r6
__fixunssfsi	float → unsigned int	%r4 ← %r6
__fixsfsi	float → int	%r4 ← %r6
__floatsisf	int → float	%r4 ← %r6
__truncdfsf2	double → float	%r4 ← (%r7, %r6)
__extendsfdf2	float → double	(%r5, %r4) ← %r6

Comparison

__fcmpd	double type	Changes %psr by (%r7, %r6) - (%r9, %r8)
__**df2	double type	Changes %psr and %r4 by (%r7, %r6) - (%r9, %r8) **=eq, ne, gt, ge, lt, le (%r4 = 1 if true, 0 if false)
__fcmps	float type	Changes %psr by %r6 - %13
__**sf2	float type	Changes %psr and %r4 by %r6 - %13 **=eq, ne, gt, ge, lt, le (%r4 = 1 if true, 0 if false)

Integer Division Library libgcc.a

Integer division

__divsi3	Signed division	%r4 ← %r6 / %r7
__udivsi3	Unsigned division	%r4 ← %r6 / %r7

Modulo operation

__modsi3	Signed operation	%r4 ← %r6 % %r7
__umodsi3	Unsigned operation	%r4 ← %r6 % %r7

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 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
-Infinity:	0xfff00000 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 0x7f7fffff
Minimum normalized number:	1.17549e-38f 0x00800000
Maximum unnormalized number:	1.17549e-38f 0x007fffff
Minimum unnormalized number:	1.40129e-45f 0x00000001
Infinity:	0x7f800000
-Infinity:	0xff800000

long long-type Emulation Library libgcc2.a

Operation

__muldi3	Multiplication	(%r5, %r4) ← (%r7, %r6) * (%r9, %r8)
__divdi3	Signed division	(%r5, %r4) ← (%r7, %r6) / (%r9, %r8)
__udivdi3	Unsigned division	(%r5, %r4) ← (%r7, %r6) / (%r9, %r8)
__moddi3	Signed remainder calculation	(%r5, %r4) ← (%r7, %r6) % (%r9, %r8)
__umoddi3	Unsigned remainder calculation	(%r5, %r4) ← (%r7, %r6) % (%r9, %r8)
__negdi2	Sign change	(%r5, %r4) ← (%r7, %r6)
__lshrdi3	Logical shift to right	(%r5, %r4) ← (%r7, %r6) >> (%r9, %r8)
__ashrdi3	Arithmetical shift to right	(%r5, %r4) ← (%r7, %r6) >> (%r9, %r8)
__ashldi3	Arithmetical shift to left	(%r5, %r4) ← (%r7, %r6) << (%r9, %r8)

Type conversion

__fixunsdfdi	double → unsigned long long	(%r5, %r4) ← (%r7, %r6)
__fixdfdi	double → long long	(%r5, %r4) ← (%r7, %r6)
__floatdidf	long long → double	(%r5, %r4) ← (%r7, %r6)
__fixunssfdi	float → unsigned long long	(%r5, %r4) ← %r6
__fixsfdi	float → long long	(%r5, %r4) ← %r6
__floatdisf	long long → float	%r4 ← (%r7, %r6)

Comparison

__cmpdi2	Comparison of long long type	Changes %psr ← (%r7, %r6) - (%r9, %r8), %r4 ← 2 | 1 | 0
__ucmpdi2	Comparison of unsigned long long type	Changes %psr ← (%r7, %r6) - (%r9, %r8), %r4 ← 2 | 1 | 0

Other

__ffsdi2	Bit scan	(%r5, %r4) ← (%r7, %r6)

Input/Output Functions (header file: stdio.h/cstdio)

fopen( )	FILE *fopen(const char *filename, const char *mode);	(dummy)*1
freopen( )	FILE *freopen(const char *filename, const char *mode, FILE *stream);	(dummy)*1
fclose( )	int fclose(FILE *stream);	(dummy)
fflush( )	int fflush(FILE *stream);	(dummy)
fseek( )	int fseek(FILE *stream, long offset, int origin);	(dummy)*1
ftell( )	long ftell(FILE *stream);	(dummy)
rewind( )	void rewind(FILE *stream);	(dummy)
fgetpos( )	int fgetpos(FILE *stream, fpos_t *ptr);	(dummy)
fsetpos( )	int fsetpos(FILE *stream, const fpost_t *pos);	(dummy)*1
fread( )	size_t fread(void *ptr, size_t size, size_t count, FILE *stream);	*1, *2
fwrite( )	size_t fwrite(const 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(void);	*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(const char *filename);	(dummy)*1
rename( )	int rename(const char *oldname, const 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(void);	(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(const char *s);	*1, *2
fscanf( )	int fscanf(FILE *stream, const char *format, ...);	*1, *2
scanf( )	int scanf(const char *format, ...);	*1, *2
sscanf( )	int sscanf(const char *s, const char *format, ...);	*1, *2
fprintf( )	int fprintf(FILE *stream, const char *format, ...);	*1, *2
printf( )	int printf(const char *format, ...);	*1, *2
sprintf( )	int sprintf(char *s, const char *format, ...);	*1, *2
vfprintf( )	int vfprintf(FILE *stream, const char *format, va_list arg);	*1, *2
vprintf( )	int vprintf(const char *format, va_list arg);	*1, *2
vsprintf( )	int vsprintf(char *s, const char *format, va_list arg);	

Utility Functions (header file: stdlib.h/cstdlib)

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(const char *command);	
exit( )	void exit(int status);	
abort( )	void abort(void);	
atexit( )	int atexit(void (*func)(void));	
getenv( )	char *getenv(const char *str);	
bsearch( )	void *bsearch(const void *key, const void *base, size_t count, size_t size, int (*compare)(const void *, const void *));	
qsort( )	void qsort(void *base, size_t count, size_t size, int (*compare)(const void *, const void *));	
abs( )	int abs(int x);	
labs( )	long labs(long x);	
div( )	div_t div(int n, int d);	*1
ldiv( )	ldiv_t ldiv(long n, long d);	*1
rand( )	int rand(void);	
srand( )	void srand(unsigned int seed);	
atol( )	long atol(const char *str);	
atoi( )	int atoi(const char *str);	*1
atof( )	double atof(const char *str);	*1
strtod( )	double strtod(const char *str, char **ptr);	*1
strtol( )	long strtol(const char *str, char **ptr, int base);	*1
strtoul( )	unsigned long strtoul(const char *str, char **ptr, int base);	*1
strtoll( )	long long strtoll(const char *str, char **endptr, int base);	*3
strtoull( )	unsigned long long strtoull(const char *str, char **endptr, int base);	*3

Date and Time Functions (header file: time.h/ctime)

clock( )	clock_t clock(void);	(dummy)
asctime( )	char *asctime(const struct tm *ts);	(dummy)
ctime( )	char *ctime(const time_t *timeptr);	(dummy)
difftime( )	double difftime(time_t ti, time_t t2);	(dummy)
gmtime( )	struct tm *gmtime(const time_t *t);	
localtime( )	struct tm *localtime(const time_t *t);	(dummy)
mktime( )	time_t mktime(struct tm *tmptr);	
time( )	time_t time(time_t *tptr);	*1

Non-local Branch Functions (header file: setjmp.h/csetjmp)

setjmp( )	int setjmp(jmp_buf env);
longjmp( )	void longjmp(jmp_buf env, int status);

*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.
*3 These utility functions included in libgcc2.a.

Mathematical Functions (header file: math.h, errno.h, float.h, limits.h/cmath, cerrno, cfloat, climits)

fabs( )	double fabs(double x);	*1
ceil( )	double ceil(double x);	*1
floor( )	double floor(double x);	*1
fmod( )	double fmod(double x, double y);	*1
exp( )	double exp(double x);	*1
log( )	double log(double x);	*1
log10( )	double log10(double x);	*1
frexp( )	double frexp(double x, int *nptr);	*1
ldexp( )	double ldexp(double x, int n);	*1
modf( )	double modf(double x, double *nptr);	*1
pow( )	double pow(double x, double y);	*1
sqrt( )	double sqrt(double x);	*1
sin( )	double sin(double x);	*1
cos( )	double cos(double x);	*1
tan( )	double tan(double x);	*1
asin( )	double asin(double x);	*1
acos( )	double acos(double x);	*1
atan( )	double atan(double x);	
atan2( )	double atan2(double y, double x);	*1
sinh( )	double sinh(double x);	*1
cosh( )	double cosh(double x);	*1
tanh( )	double tanh(double x);	

Character Type Determination/Conversion Functions (header file: ctype.h/cctype)

isalnum( )	int isalnum(int c);
isalpha( )	int isalpha(int c);
iscntrl( )	int iscntrl(int c);
isdigit( )	int isdigit(int c);
isgraph( )	int isgraph(int c);
islower( )	int islower(int c);
isprint( )	int isprint(int c);
ispunct( )	int ispunct(int c);
isspace( )	int isspace(int c);
isupper( )	int isupper(int c);
isxdigit( )	int isxdigit(int c);
tolower( )	int tolower(int c);
toupper( )	int toupper(int c);

Variable Argument Macros (header file: stdarg.h/cstdarg)

va_start( )	void va_start(va_list ap, type lastarg);
va_arg( )	type va_arg(va_list ap, type);
va_end( )	void va_end(va_list ap);

*1 These functions need to declare and initialize the global variables.

Character Functions (header file: string.h/cstring)

memchr( )	void *memchr(const void *s, int c, size_t n);
memcmp( )	int memcmp(const void *s1, const void *s2, size_t n);
memcpy( )	void *memcpy(void *s1, const void *s2, size_t n);
memmove( )	void *memmove(void *s1, const void *s2, size_t n);
memset( )	void *memset(void *s, int c, size_t n);
strcat( )	char *strcat(char *s1, const char *s2);
strchr( )	char *strchr(const char *s, int c);
strcmp( )	int strcmp(const char *s1, const char *s2);
strcpy( )	char *strcpy(char *s1, const char *s2);
strcspn( )	size_t strcspn(const char *s1, const char *s2);
strerror( )	char *strerror(int code);
strlen( )	size_t strlen(const char *s);
strncat( )	char *strncat(char *s1, const char *s2, size_t n);
strncmp( )	int strncmp(const char *s1, const char *s2, size_t n);
strncpy( )	char *strncpy(char *s1, const char *s2, size_t n);
strpbrk( )	char *strpbrk(const char *s1, const char *s2);
strrchr( )	char *strrchr(const char *str, int c);
strspn( )	size_t strspn(const char *s1, const char *s2);
strstr( )	char *strstr(const char *s1, const char *s2);
strtok( )	char *strtok(char *s1, const char *s2);

Test Macro (header file: assert.h/cassert)

assert( )	void assert(int test);

Locale Function (header file: locale.h/clocale)

setlocale( )	char *setlocale(int category, const char *locale);	(dummy)
localeconv( )	struct lconv *localeconv(void);	(dummy)

*1 Declaring and Initializing Global Variables

FILE _iob[FOPEN_MAX+1];	_iob[N]._flg=_UGETN; _iob[N]._buf=0; _iob[N]._fd=N; (N=0: stdin, N=1: stdout, N=2: stderr)
FILE *stdin;	stdin=&_iob[0];
FILE *stdout;	stdout=&_iob[1];
FILE *stderr;	stderr=&_iob[2];
int errno;	errno=0;
unsigned int seed;	seed=1;
time_t gm_sec;	gm_sec=-1;

*2 Definition of Lower-level Functions

read( )	int read(int fd, char *buf, int nbytes); unsigned char READ_BUF[65]; (Variable name is arbitrary) unsigned char READ_EOF;
write( )	int write(int fd, char *buf, int nbytes); unsigned char WRITE_BUF[65]; (Variable name is arbitrary)

STL Container Member Functions

	Container name									
Member function	deque	list	map	multimap	multiset	priority_queue	queue	set	stack	vector
assign	○	○								○
at	○									○
back	○	○					○			○
begin	○	○	○	○	○			○		○
capacity										○
clear	○	○	○	○	○			○		○
count			○	○	○			○		
empty	○	○	○	○	○	○	○	○	○	○
end	○	○	○	○	○			○		○
equal_range			○	○	○			○		
erase	○	○	○	○	○			○		○
find			○	○	○			○		
front	○	○					○			○
get_allocator	○	○	○	○	○			○		○
insert	○	○	○	○	○			○		○
key_comp			○	○	○			○		
lower_bound			○	○	○			○		
max_size	○	○	○	○	○			○		○
merge		○								
operator[ ]	○		○							○
pop						○	○		○	
pop_back	○	○								○
pop_front	○	○								
push						○	○		○	
push_back	○	○								○
push_front	○	○								
rbegin	○	○	○	○	○			○		○
remove		○								
remove_if		○								
rend	○	○	○	○	○			○		○
reserve										○
resize	○	○								○
reverse		○								
size	○	○	○	○	○	○	○	○	○	○
sort		○								
splice		○								
swap	○	○	○	○	○			○		○
top						○			○	
unique		○								
upper_bound			○	○	○			○		
value_comp			○	○	○			○		

Symbols in the Instruction List

Registers/Register Data

%rd, rd:	A general-purpose register (R0–R15) used as the destination register or the contents of the register.
%rs, rs:	A general-purpose register (R0–R15) used as the source register or the contents of the register.
%rb, rb:	A general-purpose register (R0–R15) that has stored a base address accessed in the register indirect addressing mode or the contents of the register.
%sd, sd:	A special register (PSR, SP, ALR, AHR) used as the destination register or the contents of the register.
%ss, ss:	A special register (PSR, SP, ALR, AHR) used as the source register or the contents of the register.
	* The special registers listed below can also be specified in the C33 ADV Core and C33 PE Core instruction set. C33 ADV: LCO, LSA, LEA, SOR, TTBR, DP, USP, SSP, IDIR, DBBR C33 PE: TTBR, IDIR, DBBR
%sp, sp:	Stack pointer (SP) or the contents of the pointer.
%dp, dp:	Data pointer (DP) or the contents of the pointer. (C33 ADV Core)
%rc, rc:	Loop counter (LCO) or the contents of the counter. (C33 ADV Core)
%ra, ra:	Loop end address register (LEA) or the register. (C33 ADV Core)

Memory/Addresses/Memory Data

[%rb]:	Specification for register indirect addressing.
[%rb]+:	Specification for register indirect addressing with post-increment.
[%sp+immX], [%rb+immX], [%rb+symbol±immX]:	Specification for register indirect addressing with a displacement.
B[XXX]:	The address specified with XXX, or the byte data stored in the address.
H[XXX]:	The half-word space in which the base address is specified with XXX, or the half-word data stored in the space.
W[XXX]:	The word space in which the base address is specified with XXX, or the word data stored in the space.

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
×:	Multiplication

Flags

RM:	Repeat mode enable flag (C33 ADV Core)
LM:	Loop mode enable flag (C33 ADV Core)
PM:	PUSH/POP mode flag (C33 ADV Core)
RC:	Register counter (C33 ADV Core)
S:	Saturation flag (C33 ADV Core)
ME:	MMU exception flag (C33 ADV Core)
MO:	MAC overflow flag (This flag is not supported in C33 PE Core.)
DS:	Dividend sign flag (This flag is not supported in C33 PE Core.)
C:	Carry flag
V:	Overflow flag
Z:	Zero flag
N:	Negative flag
–:	Not changed
↔:	Set (1), reset (0) or not changed
0:	Reset (0)

D

○:	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 S1C33000 instruction set and the extended instructions (x..., except for xor).
- *"Italic basic instructions"* indicate that the upper compatible extended instructions are provided.
- Instruction Lists (2) through (7) show the common instructions used for the C33 STD Core, C33 ADV Core and C33 PE Core. However, some instructions have extended functions added for the C33 ADV Core and C33 PE Core (the extended contents are described in the Remarks column).
  Instruction List (8) is dedicated for the C33 PE Core. Instruction List (9) and subsequent lists are dedicated for the C33 ADV Core, and cannot be used for the C33 STD Core.
- Some instructions have a operand including "symbol+imm26", note, however, that the valid range of values to be set is 0 ≤ offset value (symbol + imm26 - __dp) < 0x400000.

Classification	Mnemonic		Function	Flags						D
	Opcode	Operand		MO	DS	C	V	Z	N	
Signed byte data transfer	ld.b	%rd, %rs	rd(7:0)←rs(7:0), rd(31:8)←rs(7)	–	–	–	–	–	–	○
		%rd, [%rb]	rd(7:0)←B[rb], rd(31:8)←B[rb](7)	–	–	–	–	–	–	–
		%rd, [%rb]+	rd(7:0)←B[rb], rd(31:8)←B[rb](7), rb←rb+1	–	–	–	–	–	–	–
		%rd, [%sp+imm6]	rd(7:0)←B[sp+imm6], rd(31:8)←B[sp+imm6](7)	–	–	–	–	–	–	–
		[%rb], %rs	B[rb]←rs(7:0)	–	–	–	–	–	–	–
		[%rb]+, %rs	B[rb]←rs(7:0), rb←rb+1	–	–	–	–	–	–	–
		[%sp+imm6], %rs	B[sp+imm6]←rs(7:0)	–	–	–	–	–	–	–
	xld.b	%rd, [symbol+imm26]	rd(7:0)←B[symbol+imm26], rd(31:8)←B[symbol+imm26](7) (∗1)	–	–	–	–	–	–	–
		%rd, [%rb+imm26]	rd(7:0)←B[rb+imm26], rd(31:8)←B[rb+imm26](7)	–	–	–	–	–	–	–
		%rd, [%sp+imm32]	rd(7:0)←B[sp+imm32], rd(31:8)←B[sp+imm32](7)	–	–	–	–	–	–	–
		[symbol+imm26], %rs	B[symbol+imm26]←rs(7:0) (∗1)	–	–	–	–	–	–	–
		[%rb+imm26], %rs	B[rb+imm26]←rs(7:0)	–	–	–	–	–	–	–
		[%sp+imm32], %rs	B[sp+imm32]←rs(7:0)	–	–	–	–	–	–	–
Unsigned byte data transfer	ld.ub	%rd, %rs	rd(7:0)←rs(7:0), rd(31:8)←0	–	–	–	–	–	–	○
		%rd, [%rb]	rd(7:0)←B[rb], rd(31:8)←0	–	–	–	–	–	–	–
		%rd, [%rb]+	rd(7:0)←B[rb], rd(31:8)←0, rb←rb+1	–	–	–	–	–	–	–
		%rd, [%sp+imm6]	rd(7:0)←B[sp+imm6], rd(31:8)←0	–	–	–	–	–	–	–
	xld.ub	%rd, [symbol+imm26]	rd(7:0)←B[symbol+imm26], rd(31:8)←0 (∗1)	–	–	–	–	–	–	–
		%rd, [%rb+imm26]	rd(7:0)←B[rb+imm26], rd(31:8)←0	–	–	–	–	–	–	–
		%rd, [%sp+imm32]	rd(7:0)←B[sp+imm32], rd(31:8)←0	–	–	–	–	–	–	–
Signed half word data transfer	ld.h	%rd, %rs	rd(15:0)←rs(15:0), rd(31:16)←rs(15)	–	–	–	–	–	–	○
		%rd, [%rb]	rd(15:0)←H[rb], rd(31:16)←H[rb](15)	–	–	–	–	–	–	–
		%rd, [%rb]+	rd(15:0)←H[rb], rd(31:16)←H[rb](15), rb←rb+2	–	–	–	–	–	–	–
		%rd, [%sp+imm6]	rd(15:0)←H[sp+imm7], rd(31:16)←H[sp+imm7](15); imm7={imm6,0}	–	–	–	–	–	–	–
		[%rb], %rs	H[rb]←rs(15:0)	–	–	–	–	–	–	–
		[%rb]+, %rs	H[rb]←rs(15:0), rb←rb+2	–	–	–	–	–	–	–
		[%sp+imm6], %rs	H[sp+imm7]←rs(15:0); imm7={imm6,0}	–	–	–	–	–	–	–
	xld.h	%rd, [symbol+imm26]	rd(15:0)←H[symbol+imm26], rd(31:16)←H[symbol+imm26](15) (∗1)	–	–	–	–	–	–	–
		%rd, [%rb+imm26]	rd(15:0)←H[rb+imm26], rd(31:16)←H[rb+imm26](15)	–	–	–	–	–	–	–
		%rd, [%sp+imm32]	rd(15:0)←H[sp+imm32], rd(31:16)←H[sp+imm32](15)	–	–	–	–	–	–	–
		[symbol+imm26], %rs	H[symbol+imm26]←rs(15:0) (∗1)	–	–	–	–	–	–	–
		[%rb+imm26], %rs	H[rb+imm26]←rs(15:0)	–	–	–	–	–	–	–
		[%sp+imm32], %rs	H[sp+imm32]←rs(15:0)	–	–	–	–	–	–	–
Unsigned half word data transfer	ld.uh	%rd, %rs	rd(15:0)←rs(15:0), rd(31:16)←0	–	–	–	–	–	–	○
		%rd, [%rb]	rd(15:0)←H[rb], rd(31:16)←0	–	–	–	–	–	–	–
		%rd, [%rb]+	rd(15:0)←H[rb], rd(31:16)←0, rb←rb+2	–	–	–	–	–	–	–
		%rd, [%sp+imm6]	rd(15:0)←H[sp+imm7], rd(31:16)←0; imm7={imm6,0}	–	–	–	–	–	–	–

Remarks

∗1) In the C33 ADV Core, imm26 is extended into imm32.

Classification	Mnemonic		Function	Flags						D
	Opcode	Operand		MO	DS	C	V	Z	N	
Unsigned half word data transfer	xld.uh	%rd, [symbol+imm26]	rd(15:0)←H[symbol+imm26], rd(31:16)←0 (∗1)	–	–	–	–	–	–	–
		%rd, [%rb+imm26]	rd(15:0)←H[rb+imm26], rd(31:16)←0	–	–	–	–	–	–	–
		%rd, [%sp+imm32]	rd(15:0)←H[sp+imm32], rd(31:16)←0	–	–	–	–	–	–	–
Word data transfer	ld.w	%rd, %rs	rd←rs	–	–	–	–	–	–	○
		%sd, %rs	sd←rs	–	–	–	–	–	–	–
		%rd, %ss	rd←ss	–	–	–	–	–	–	–
		%rd, sign6	rd(5:0)←sign6(5:0), rd(31:6)←sign6(5)	–	–	–	–	–	–	○
		%rd, [%rb]	rd←W[rb]	–	–	–	–	–	–	–
		%rd, [%rb]+	rd←W[rb], rb←rb+4	–	–	–	–	–	–	–
		%rd, [%sp+imm6]	rd←W[sp+imm8]; imm8={imm6,00}	–	–	–	–	–	–	–
		[%rb], %rs	W[rb]←rs	–	–	–	–	–	–	–
		[%rb]+, %rs	W[rb]←rs, rb←rb+4	–	–	–	–	–	–	–
		[%sp+imm6], %rs	W[sp+imm8]←rs; imm8={imm6,00}	–	–	–	–	–	–	–
	xld.w	%rd, sign32	rd←sign32	–	–	–	–	–	–	∗3
		%rd, symbol+imm32	rd←symbol+imm32	–	–	–	–	–	–	–
		%rd, symbol-imm32	rd←symbol-imm32	–	–	–	–	–	–	–
		%rd, [symbol+imm26]	rd←W[symbol+imm26] (∗1)	–	–	–	–	–	–	–
		%rd, [%rb+imm26]	rd←W[rb+imm26]	–	–	–	–	–	–	–
		%rd, [%sp+imm32]	rd←W[sp+imm32]	–	–	–	–	–	–	–
		[symbol+imm26], %rs	W[symbol+imm26]←rs (∗1)	–	–	–	–	–	–	–
		[%rb+imm26], %rs	W[rb+imm26]←rs	–	–	–	–	–	–	–
		[%sp+imm32], %rs	W[sp+imm32]←rs	–	–	–	–	–	–	–
Logic operation	and	%rd, %rs	rd←rd & rs (∗2)	–	–	–	–	↔	↔	○
		%rd, sign6	rd←rd & sign6(with sign extension) (∗2)	–	–	–	–	↔	↔	∗3
	xand	%rd, sign32	rd←rd & sign32	–	–	–	–	↔	↔	∗3
	or	%rd, %rs	rd←rd | rs (∗2)	–	–	–	–	↔	↔	○
		%rd, sign6	rd←rd | sign6(with sign extension) (∗2)	–	–	–	–	↔	↔	∗3
	xoor	%rd, sign32	rd←rd | sign32	–	–	–	–	↔	↔	∗3
	xor	%rd, %rs	rd←rd ^ rs (∗2)	–	–	–	–	↔	↔	○
		%rd, sign6	rd←rd ^ sign6(with sign extension) (∗2)	–	–	–	–	↔	↔	∗3
	xxor	%rd, sign32	rd←rd ^ sign32	–	–	–	–	↔	↔	∗3
	not	%rd, %rs	rd←!rs (∗2)	–	–	–	–	↔	↔	○
		%rd, sign6	rd←!sign6(with sign extension) (∗2)	–	–	–	–	↔	↔	∗3
	xnot	%rd, sign32	rd←!sign32	–	–	–	–	↔	↔	∗3

Remarks

∗1) In the C33 ADV Core, imm26 is extended into imm32.

∗2) In the C33 ADV Core and when the OC flag is set to "1", the V flag is cleared after the instruction is executed. In the C33 PE Core, the V flag is cleared after the instruction is executed.

∗3) Can be used as a delayed instruction only when the immediate operand has a value within the range for sign6.

Classification	Mnemonic		Function	Flags						D
	Opcode	Operand		MO	DS	C	V	Z	N	
Arithmetic operation	add	%rd, %rs	rd←rd + rs	–	–	↔	↔	↔	↔	○
		%rd, imm6	rd←rd + imm6(with zero extension)	–	–	↔	↔	↔	↔	○
		%sp, imm10	sp←sp + imm12(with zero extension); imm12={imm10,00}	–	–	–	–	–	–	○
	xadd	%rd, imm32	rd←rd + imm32	–	–	↔	↔	↔	↔	∗1
	adc	%rd, %rs	rd←rd + rs + C	–	–	↔	↔	↔	↔	○
	sub	%rd, %rs	rd←rd - rs	–	–	↔	↔	↔	↔	○
		%rd, imm6	rd←rd - imm6(with zero extension)	–	–	↔	↔	↔	↔	○
		%sp, imm10	sp←sp - imm12(with zero extension); imm12={imm10,00}	–	–	–	–	–	–	○
	xsub	%rd, imm32	rd←rd - imm32	–	–	↔	↔	↔	↔	∗1
	sbc	%rd, %rs	rd←rd - rs - C	–	–	↔	↔	↔	↔	○
	cmp	%rd, %rs	rd - rs	–	–	↔	↔	↔	↔	○
		%rd, sign6	rd - sign6(with sign extension)	–	–	↔	↔	↔	↔	○
	xcmp	%rd, sign32	rd - sign32	–	–	↔	↔	↔	↔	∗1
	mlt.h	%rd, %rs	alr←rd(15:0) × rs(15:0); calculated with sign	–	–	–	–	–	–	○
	mltu.h	%rd, %rs	alr←rd(15:0) × rs(15:0); calculated without sign	–	–	–	–	–	–	○
	mlt.w	%rd, %rs	{ahr, alr}←rd × rs; calculated with sign	–	–	–	–	–	–	–
	mltu.w	%rd, %rs	{ahr, alr}←rd × rs; calculated without sign	–	–	–	–	–	–	–
	div0s (∗6)	%rs	Setup for signed division; alr = dividend, rs = divisor	–	↔	–	–	–	↔	–
	div0u (∗6)	%rs	Setup for unsigned division; alr = dividend, rs = divisor	–	0	–	–	–	0	–
	div1 (∗6)	%rs	Step division for one bit (∗2); alr←quotient, ahr←remainder (unsigned)	–	–	–	–	–	–	–
	div2s (∗6)	%rs	Correction step 1 for signed division (∗3)	–	–	–	–	–	–	–
	div3s (∗6)		Correction step 2 for signed division (∗3); alr←quotient, ahr←remainder	–	–	–	–	–	–	–
Shift & rotation	*srl*	*%rd, imm4*	Logical shift to right imm4 bits; imm4=0–8, zero enters to MSB (∗4)	–	–	–	–	↔	↔	○
		%rd, %rs	Logical shift to right rs bits; rs=0–8, zero enters to MSB (∗4)	–	–	–	–	↔	↔	○
	xsrl	%rd, imm5	Logical shift to right imm5 bits; imm5=0–31, zero enters to MSB	–	–	–	–	↔	↔	∗5
	sll	*%rd, imm4*	Logical shift to left imm4 bits; imm4=0–8, zero enters to LSB (∗4)	–	–	–	–	↔	↔	○
		%rd, %rs	Logical shift to left rs bits; rs=0–8, zero enters to LSB (∗4)	–	–	–	–	↔	↔	○
	xsll	%rd, imm5	Logical shift to left imm5 bits; imm5=0–31, zero enters to LSB	–	–	–	–	↔	↔	∗5
	sra	*%rd, imm4*	Arithmetical shift to right imm4 bits; imm4=0–8, sign copied to MSB (∗4)	–	–	–	–	↔	↔	○
		%rd, %rs	Arithmetical shift to right rs bits; rs=0–8, sign copied to MSB (∗4)	–	–	–	–	↔	↔	○
	xsra	%rd, imm5	Arithmetical shift to right imm5 bits; imm5=0–31, sign copied to MSB	–	–	–	–	↔	↔	∗5

Remarks

∗1) Can be used as a delayed instruction only when the immediate operand has a value within the range for sign6.

∗2) The div1 instruction must be executed 32 times when performing 32-bit data ÷ 32-bit data. In unsigned division, the division result is loaded to the alr and ahr registers.

∗3) It is not necessary to execute the div2s and div3s instructions for unsigned division.

∗4) In the C33 ADV Core and C33 PE Core, "imm4=0–8" and "rs=0–8" are extended into "imm5=0–31" and "rs=0–31", respectively. Furthermore, in the C33 ADV Core, the C flag changes according to the results when the SE flag is set to "1".

∗5) Can be used as a delayed instruction only in the C33 ADV Core.

∗6) The instruction is not supported in the C33 PE Core. If these instructions are executed, the instruction code will be stored in 16 low order-bits of the IDIR register, and an undefined instruction exception (TTBR + 12) will occur.

Classification	Mnemonic		Function	Flags						D
	Opcode	Operand		MO	DS	C	V	Z	N	
Shift & rotation	*sla*	*%rd, imm4*	Arithmetical shift to left imm4 bits; imm4=0–8, zero enters to LSB (∗1)	–	–	–	–	↔	↔	○
		%rd, %rs	Arithmetical shift to left rs bits; rs=0–8, zero enters to LSB (∗1)	–	–	–	–	↔	↔	○
	xsla	%rd, imm5	Arithmetical shift to left imm5 bits; imm5=0–31, zero enters to LSB	–	–	–	–	↔	↔	∗2
	rr	*%rd, imm4*	Rotation to right imm4 bits; imm4=0–8, LSB goes to MSB (∗1)	–	–	–	–	↔	↔	○
		%rd, %rs	Rotation to right rs bits; rs=0–8, LSB goes to MSB (∗1)	–	–	–	–	↔	↔	○
	xrr	%rd, imm5	Rotation to right imm5 bits; imm5=0–31, LSB goes to MSB	–	–	–	–	↔	↔	∗2
	rl	*%rd, imm4*	Rotation to left imm4 bits; imm4=0–8, MSB goes to LSB (∗1)	–	–	–	–	↔	↔	○
		%rd, %rs	Rotation to left rs bits; rs=0–8, MSB goes to LSB (∗1)	–	–	–	–	↔	↔	○
	xrl	%rd, imm5	Rotation to left imm5 bits; imm5=0–31, MSB goes to LSB	–	–	–	–	↔	↔	∗2
Bit operation	*btst*	*[%rb], imm3*	Z flag←1 if B[rb](imm3)=0	–	–	–	–	↔	–	–
	xbtst	[symbol+imm26], imm3	Z flag←1 if B[symbol+imm26](imm3)=0	–	–	–	–	↔	–	–
		[%rb+imm26], imm3	Z flag←1 if B[rb+imm26](imm3)=0	–	–	–	–	↔	–	–
	bclr	*[%rb], imm3*	B[imm32](imm3)←0	–	–	–	–	–	–	–
	xbclr	[symbol+imm26], imm3	B[symbol+imm26](imm3)←0	–	–	–	–	–	–	–
		[%rb+imm26], imm3	B[rb+imm26](imm3)←0	–	–	–	–	–	–	–
	bset	*[%rb], imm3*	B[rb](imm3)←1	–	–	–	–	–	–	–
	xbset	[symbol+imm26], imm3	B[symbol+imm26](imm3)←1	–	–	–	–	–	–	–
		[%rb+imm26], imm3	B[rb+imm26](imm3)←1	–	–	–	–	–	–	–
	bnot	*[%rb], imm3*	B[rb](imm3)←!B[rb](imm3)	–	–	–	–	–	–	–
	xbnot	[symbol+imm26], imm3	B[symbol+imm26](imm3)←!B[symbol+imm26](imm3)	–	–	–	–	–	–	–
		[%rb+imm26], imm3	B[rb+imm26](imm3)←!B[rb+imm26](imm3)	–	–	–	–	–	–	–
Branch	*jrgt* *jrgt.d*	*sign8*	pc←pc+sign9 if !Z&!(N^V) is true; sign9={sign8,0}	–	–	–	–	–	–	–
	xjrgt	label+imm32	pc←label+imm32 if !Z&!(N^V) is true	–	–	–	–	–	–	–
	xjrgt.d	sign32	pc←pc+sign32 if !Z&!(N^V) is true	–	–	–	–	–	–	–
	jrge *irge.d*	*sign8*	pc←pc+sign9 if !(N^V) is true; sign9={sign8,0}	–	–	–	–	–	–	–
	xjrge	label+imm32	pc←label+imm32 if !(N^V) is true	–	–	–	–	–	–	–
	xirge.d	sign32	pc←pc+sign32 if !(N^V) is true	–	–	–	–	–	–	–
	jrlt *jrlt.d*	*sign8*	pc←pc+sign9 if N^V is true; sign9={sign8,0}	–	–	–	–	–	–	–
	xjrlt	label+imm32	pc←label+imm32 if N^V is true	–	–	–	–	–	–	–
	xjrlt.d	sign32	pc←pc+sign32 if N^V is true	–	–	–	–	–	–	–

Remarks

∗1) In the C33 ADV Core and C33 PE Core, "imm4=0–8" and "rs=0–8" are extended into "imm5=0–31" and "rs=0–31", respectively. Furthermore, in the C33 ADV Core, the C flag changes according to the results when the SE flag is set to "1".

∗2) Can be used as a delayed instruction only in the C33 ADV Core.

Classification	Mnemonic		Function	Flags						D
	Opcode	Operand		MO	DS	C	V	Z	N	
Branch	*jrle* *jrle.d*	*sign8*	pc←pc+sign9 if Z | (N^V) is true; sign9={sign8,0}	–	–	–	–	–	–	–
	xjrle	label+imm32	pc←label+imm32 if Z | (N^V) is true	–	–	–	–	–	–	–
	xjrle.d	sign32	pc←pc+sign32 if Z | (N^V) is true	–	–	–	–	–	–	–
	jrugt *jrugt.d*	*sign8*	pc←pc+sign9 if !Z&!C is true; sign9={sign8,0}	–	–	–	–	–	–	–
	xjrugt	label+imm32	pc←label+imm32 if !Z&!C is true	–	–	–	–	–	–	–
	xjrugt.d	sign32	pc←pc+sign32 if !Z&!C is true	–	–	–	–	–	–	–
	jruge *jruge.d*	*sign8*	pc←pc+sign9 if !C is true; sign9={sign8,0}	–	–	–	–	–	–	–
	xjruge	label+imm32	pc←label+imm32 if !C is true	–	–	–	–	–	–	–
	xjruge.d	sign32	pc←pc+sign32 if !C is true	–	–	–	–	–	–	–
	jrult *jrult.d*	*sign8*	pc←pc+sign9 if C is true; sign9={sign8,0}	–	–	–	–	–	–	–
	xjrult	label+imm32	pc←label+imm32 if C is true	–	–	–	–	–	–	–
	xjrult.d	sign32	pc←pc+sign32 if C is true	–	–	–	–	–	–	–
	jrule *jrule.d*	*sign8*	pc←pc+sign9 if Z | C is true; sign9={sign8,0}	–	–	–	–	–	–	–
	xjrule	label+imm32	pc←label+imm32 if Z | C is true	–	–	–	–	–	–	–
	xjrule.d	sign32	pc←pc+sign32 if Z | C is true	–	–	–	–	–	–	–
	jreq *jreq.d*	*sign8*	pc←pc+sign9 if Z is true; sign9={sign8,0}	–	–	–	–	–	–	–
	xjreq	label+imm32	pc←label+imm32 if Z is true	–	–	–	–	–	–	–
	xjreq.d	sign32	pc←pc+sign32 if Z is true	–	–	–	–	–	–	–
	jrne *jrne.d*	*sign8*	pc←pc+sign9 if !Z is true; sign9={sign8,0}	–	–	–	–	–	–	–
	xjrne	label+imm32	pc←label+imm32 if !Z is true	–	–	–	–	–	–	–
	xjrne.d	sign32	pc←pc+sign32 if !Z is true	–	–	–	–	–	–	–
	call	*sign8*	sp←sp-4, W[sp]←pc+2, pc←pc+sign9; sign9={sign8,0}	–	–	–	–	–	–	–
	call.d	%rb	sp←sp-4, W[sp]←pc+2, pc←rb	–	–	–	–	–		–
	xcall	label+imm32	sp←sp-4, W[sp]←pc+2, pc←label+imm32	–	–	–	–	–	–	–
	xcall.d	sign32	sp←sp-4, W[sp]←pc+2, pc←pc+sign32	–	–	–	–	–	–	–
	jp	*sign8*	pc←pc+sign9; sign9={sign8,0}	–	–	–	–	–	–	–
	jp.d	%rb	pc←rb	–	–	–	–	–	–	–
	xjp	label+imm32	pc←pc+label+imm32	–	–	–	–	–	–	–
	xjp.d	sign32	pc←pc+sign32	–	–	–	–	–	–	–

Remarks

Instruction List (7)

Classification	Mnemonic		Function	Flags						D
	Opcode	Operand		MO	DS	C	V	Z	N	
Branch	ret ret.d		pc←W[sp], sp←sp+4	–	–	–	–	–	–	–
	reti		psr←W[sp], sp←sp+4, pc←W[sp], sp←sp+4	↔	↔	↔	↔	↔	↔	–
	retd		Returns from debugging routine (for S5U1C33xxxH software)	–	–	–	–	–	–	–
	int	imm2	sp←sp-4, W[sp]←pc+2, sp←sp-4, W[sp]←psr, pc←software exception vector	–	–	–	–	–	–	–
	brk		Interrupt for debugging (for S5U1C33xxxH software)	–	–	–	–	–	–	–
Extension	*ext*	*imm13*	Extends the immediate or operand of the following instruction.	–	–	–	–	–	–	–
Push & pop	pushn	%rs	Repeats "sp←sp-4, W[sp]←rn"; rn=rs to r0 (∗1)	–	–	–	–	–	–	–
	popn	%rd	Repeats "rn←W[sp], sp←sp+4"; rn=r0 to rd (∗1)	–	–	–	–	–	–	–
MAC	mac (∗4)	%rs	Repeats "{ahr, alr}←{ahr, alr} + H[<rs+1>]+ × H[<rs+2>]+" rs times (∗2)	↔	–	–	–	–	–	–
System control	nop		No operation; pc←pc+2	–	–	–	–	–	–	–
	halt		Sets Halt mode	–	–	–	–	–	–	–
	slp		Sets Sleep mode	–	–	–	–	–	–	–
Others	scan0 (∗4)	%rd, %rs	Scan 0 bit for 1 byte from MSB in rs, rd←offset from MSB of found bit (∗3)	–	–	↔	0	↔	0	○
	scan1 (∗4)	%rd, %rs	Scan 1 bit for 1 byte from MSB in rs, rd←offset from MSB of found bit (∗3)	–	–	↔	0	↔	0	○
	swap	%rd, %rs	rd(31:24)←rs(7:0), rd(23:16)←rs(15:8), rd(15:8)←rs(23:16), rd(7:0)←rs(31:24)	–	–	–	–	–	–	○
	mirror (∗4)	%rd, %rs	rd(31:24)←rs(24:31), rd(23:16)←rs(16:23), rd(15:8)←rs(8:15), rd(7:0)←rs(0:7)	–	–	–	–	–	–	○

Remarks

∗1) In the C33 ADV Core and when the PM flag is set to "1", the register push/pop operation starts from the register number stored in RC[3:0]. The PM flag and RC[3:0] change after execution.

∗2) <rs+1>, <rs+2>: contents of the registers that follow rs. (eg. rs=r0: <rs+1>=r1, <rs+2>=r2; rs=r15: <rs+1>=r0, <rs+2>=r1); They are incremented (+2) after each operation. The mac instruction can be executed only in the models that have an optional multiplier.

∗3) In the C33 ADV Core and when the SW flag is set to "1", all the bits in rs (32 bits) are scanned.

∗4) The instruction is not supported in the C33 PE Core. If these instructions are executed, the instruction code will be stored in 16 low order-bits of the IDIR register, and an undefined instruction exception (TTBR + 12) will occur.

Instruction List (8) – C33 PE Core

Classification	Mnemonic		Function	Flags				D
	Opcode	Operand		C	V	Z	N	
Branch	jpr jpr.d	%rb	pc←pc + rb	–	–	–	–	–
System control	psrset	imm5	psr(imm5)←1	↔	↔	↔	↔	–
	psrclr	imm5	psr(imm5)←0	↔	↔	↔	↔	–
Push & pop	push	%rs	sp←sp-4, W[sp]←rs	–	–	–	–	–
	pushs	%ss	sp←sp-4, W[sp]←ahr, sp←sp-4, W[sp]←alr (if ss=ahr) sp←sp-4, W[sp]←alr (if ss=alr)	–	–	–	–	–
	pop	%rd	rd←W[sp], sp←sp+4	–	–	–	–	–
	pops	%sd	alr←W[sp], sp←sp+4, ahr←W[sp], sp←sp+4 (if sd=ahr) alr←W[sp], sp←sp+4 (if sd=alr)	–	–	–	–	–
Swap	swaph	%rd, %rs	rd(31:24)←rs(23:16), rd(23:16)←rs(31:24), rd(15:8)←rs(7:0), rd(7:0)←rs(15:8)	–	–	–	–	–
Coprocessor	ld.c	%rd, imm4	rd(7:0)←coprocessor register(imm4)	–	–	–	–	–
		imm4, %rs	Coprocessor register(imm4)←rs(7:0)	–	–	–	–	–
	do.c	imm6	Issues a command (imm6) to the coprocessor	–	–	–	–	–
	ld.cf		psr(3:0)←coprocessor flag	↔	↔	↔	↔	–

Remarks

Instruction List (9) – C33 ADV Core

Classification	Mnemonic		Function	Flags												D
	Opcode	Operand		RM	LM	PM	RC	S	ME	MO	DS	C	V	Z	N	
Signed byte data transfer	ld.b	*%rd, [%dp+imm6]*	rd(7:0)←B[dp+imm6], rd(31:8)←B[dp+imm6](7)	–	–	–	–	–	–	–	–	–	–	–	–	–
		[%dp+imm6], %rs	B[dp+imm6]←rs(7:0)	–	–	–	–	–	–	–	–	–	–	–	–	–
	ald.b	%rd, [symbol+imm19]	rd(7:0)←B[symbol+imm19], rd(31:8)←B[symbol+imm19](7)	–	–	–	–	–	–	–	–	–	–	–	–	–
		[symbol+imm19], %rs	B[symbol+imm19]←rs(7:0)	–	–	–	–	–	–	–	–	–	–	–	–	–
	xld.b	%rd, [%dp+imm32]	rd(7:0)←B[dp+imm32], rd(31:8)←B[dp+imm32](7)	–	–	–	–	–	–	–	–	–	–	–	–	–
		[%dp+imm32], %rs	B[dp+imm32]←rs(7:0)	–	–	–	–	–	–	–	–	–	–	–	–	–
Unsigned byte data transfer	ld.ub	*%rd, [%dp+imm6]*	rd(7:0)←B[dp+imm6], rd(31:8)←0	–	–	–	–	–	–	–	–	–	–	–	–	–
	ald.ub	%rd, [symbol+imm19]	rd(7:0)←B[symbol+imm19], rd(31:8)←0	–	–	–	–	–	–	–	–	–	–	–	–	–
	xld.ub	%rd, [%dp+imm32]	rd(7:0)←B[dp+imm32], rd(31:8)←0	–	–	–	–	–	–	–	–	–	–	–	–	–
Signed half word data transfer	ld.h	*%rd, [%dp+imm6]*	rd(15:0)←H[dp+imm7], rd(31:16)←H[dp+imm7](15); imm7={imm6,0}	–	–	–	–	–	–	–	–	–	–	–	–	–
		[%dp+imm6], %rs	H[dp+imm7]←rs(15:0); imm7={imm6,0}	–	–	–	–	–	–	–	–	–	–	–	–	–
	ald.h	%rd, [symbol+imm19]	rd(15:0)←H[symbol+imm19], rd(31:16)←H[symbol+imm19](15)	–	–	–	–	–	–	–	–	–	–	–	–	–
		[symbol+imm19], %rs	H[symbol+imm19]←rs(15:0)	–	–	–	–	–	–	–	–	–	–	–	–	–
	xld.h	%rd, [%dp+imm32]	rd(15:0)←H[dp+imm32], rd(31:16)←H[dp+imm32](15)	–	–	–	–	–	–	–	–	–	–	–	–	–
		[%dp+imm32], %rs	H[dp+imm32]←rs(15:0)	–	–	–	–	–	–	–	–	–	–	–	–	–
Unsigned half word data transfer	ld.uh	*%rd, [%dp+imm6]*	rd(15:0)←H[dp+imm7], rd(31:16)←0; imm7={imm6,0}	–	–	–	–	–	–	–	–	–	–	–	–	–
	ald.uh	%rd, [symbol+imm19]	rd(15:0)←H[symbol+imm19], rd(31:16)←0	–	–	–	–	–	–	–	–	–	–	–	–	–
	xld.uh	%rd, [%dp+imm32]	rd(15:0)←H[dp+imm32], rd(31:16)←0	–	–	–	–	–	–	–	–	–	–	–	–	–
Word data transfer	ld.w	*%rd, [%dp+imm6]*	rd←W[dp+imm8]; imm8={imm6,00}	–	–	–	–	–	–	–	–	–	–	–	–	–
		[%dp+imm6], %rs	W[dp+imm8]←rs; imm8={imm6,00}	–	–	–	–	–	–	–	–	–	–	–	–	–
	ald.w	%rd, [symbol+imm19]	rd←W[symbol+imm19]	–	–	–	–	–	–	–	–	–	–	–	–	–
		[symbol+imm19], %rs	W[symbol+imm19]←rs	–	–	–	–	–	–	–	–	–	–	–	–	–
	xld.w	%rd, [%dp+imm32]	rd←W[dp+imm32]	–	–	–	–	–	–	–	–	–	–	–	–	–
		[%dp+imm32], %rs	W[dp+imm32]←rs	–	–	–	–	–	–	–	–	–	–	–	–	–
Arithmetic operation	add	%rd, %dp	rd←rd + dp	–	–	–	–	–	–	–	–	–	–	–	–	○
	mlt.hw	%rd, %rs	{ahr, alr}←rd(31:0) × rs(15:0); calculated with sign	–	–	–	–	–	–	–	–	–	–	–	–	–
	div.w	%rs	alr←alr/rs(quotient), ahr←alr/rs(remainder); signed division	–	–	–	–	–	–	–	↔	–	–	–	↔	–
	divu.w	%rs	alr←alr/rs(quotient), ahr←alr/rs(remainder); unsigned division	–	–	–	–	–	–	–	0	–	–	–	0	–
Branch	jpr jpr.d	%rb	pc←pc + rb	–	–	–	–	–	–	–	–	–	–	–	–	–
	retm		r0←W[0x1C], pc←W[0x18]	–	–	–	–	–	0	–	–	–	–	–	–	–
System control	psrset	imm5	psr(imm5)←1	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	–
	psrclr	imm5	psr(imm5)←0	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	–

Remarks

Classification	Mnemonic		Function	Flags												D
	Opcode	Operand		RM	LM	PM	RC	S	ME	MO	DS	C	V	Z	N	
Extension	ext	%rs, op, imm2	3-operand execution with imm2-bit post-shift; op=sll, srl, sra	–	–	–	–	–	–	–	–	–	–	–	–	–
		op, imm2	imm2-bit post-shift; op=sll, srl, sra	–	–	–	–	–	–	–	–	–	–	–	–	–
		%rs	Executes the immediately following instruction after expanding it into 3 operands	–	–	–	–	–	–	–	–	–	–	–	–	–
		cond	Conditional execution; cond=gt, ge, lt, le, ugt, uge, ult, ule, eq, ne	–	–	–	–	–	–	–	–	–	–	–	–	–
Push & pop	push	%rs	sp←sp-4, W[sp]←rs	–	–	–	–	–	–	–	–	–	–	–	–	–
	pushs	%ss	sp←sp-4, W[sp]←ss (if ss=psr, sp) Repeats "sp←sp-4, W[sp]←sn"; sn=ss to alr (if ss=alr–pc) (∗1)	–	–	↔	↔	–	–	–	–	–	–	–	–	–
	pop	%rd	rd←W[sp], sp←sp+4	–	–	–	–	–	–	–	–	–	–	–	–	–
	pops	%sd	sd←W[sp], sp←sp+4 (if sd=psr, sp) (sd==psr)→ Repeats "sn←W[sp], sp←sp+4"; sn=alr to sd (if sd=alr–pc) (∗1)	↔ –	↔ –	↔ ↔	↔ ↔	↔ –	↔ –	↔ –	↔ –	↔ –	↔ –	↔ –	↔ –	–
MAC	mac.hw	%rs	Repeats "{ahr, alr}←{ahr, alr} + W[<rs+1>]+ × H[<rs+2>]+" rs times (∗2)	–	–	–	–	–	–	↔	–	–	–	–	–	–
	mac.w	%rs	Repeats "{ahr, alr}←{ahr, alr} + W[<rs+1>]+ × W[<rs+2>]+" rs times (∗2)	–	–	–	–	–	–	↔	–	–	–	–	–	–
	mac1.h	%rd, %rs	{ahr, alr}←{ahr, alr} + rd(15:0) × rs(15:0)	–	–	–	–	–	–	↔	–	–	–	–	–	–
	mac1.hw	%rd, %rs	{ahr, alr}←{ahr, alr} + rd(31:0) × rs(15:0)	–	–	–	–	–	–	↔	–	–	–	–	–	–
	mac1.w	%rd, %rs	{ahr, alr}←{ahr, alr} + rd(31:0) × rs(31:0)	–	–	–	–	–	–	↔	–	–	–	–	–	–
	macclr		{ahr, alr}←0, MO←0	–	–	–	–	–	–	0	–	–	–	–	–	–
Data transfer with saturation process	sat.b	%rd, %rs	rd←rs(b) if -128≤rs≤127, rd←0xFFFFFF80 if rs<-128, rd←0x0000007F if rs>127	–	–	–	–	↔	–	–	–	–	–	–	–	–
	sat.ub	%rd, %rs	rd←rs(ub) if rs≤255, rd←0x000000FF if rs>255	–	–	–	–	↔	–	–	–	–	–	–	–	–
	sat.h	%rd, %rs	rd←rs(h) if -32768≤rs≤32767, rd←0xFFFF8000 if rs<-32768, rd←0x00007FFF if rs>32767	–	–	–	–	↔	–	–	–	–	–	–	–	–
	sat.uh	%rd, %rs	rd←rs(uh) if rs≤65535, rd←0x0000FFFF if rs>65535	–	–	–	–	↔	–	–	–	–	–	–	–	–
	sat.w	%rd, %rs	rd←0x80000000 if V=1&N=0, rd←0x7FFFFFFF if V=1&N=1, otherwise rd←rs	–	–	–	–	↔	–	–	–	–	–	–	–	–
	sat.uw	%rd, %rs	rd←0xFFFFFFFF if C=1, otherwise rd←rs	–	–	–	–	↔	–	–	–	–	–	–	–	–
Loop & repeat	loop	%rc, %ra	Executes the instructions from pc+2 to ra, rc+1 time	–	↔	–	–	–	–	–	–	–	–	–	–	–
		%rc, imm4	Executes the instructions from pc+2 to pc+2+(imm4×2), rc+1 time	–	↔	–	–	–	–	–	–	–	–	–	–	–
		imm4', imm4	Executes the instructions from pc+2 to pc+2+(imm4×2), imm4'+1 time	–	↔	–	–	–	–	–	–	–	–	–	–	–
	repeat	%rc	Executes the instruction at pc+2, rc+1 time	↔	–	–	–	–	–	–	–	–	–	–	–	–
		imm4	Executes the instruction at pc+2, imm4+1 time	↔	–	–	–	–	–	–	–	–	–	–	–	–
Swap	swaph	%rd, %rs	rd(31:24)←rs(23:16), rd(23:16)←rs(31:24), rd(15:8)←rs(7:0), rd(7:0)←rs(15:8)	–	–	–	–	–	–	–	–	–	–	–	–	–
Coprocessor	ld.c	%rd, imm4	rd(7:0)←coprocessor register(imm4)	–	–	–	–	–	–	–	–	–	–	–	–	–
		imm4, %rs	Coprocessor register(imm4)←rs(7:0)	–	–	–	–	–	–	–	–	–	–	–	–	–
	do.c	imm6	Issues a command (imm6) to the coprocessor	–	–	–	–	–	–	–	–	–	–	–	–	–
	ld.cf		psr(3:0)←coprocessor flag	–	–	–	–	–	–	–	–	↔	↔	↔	↔	–

Remarks

∗1) When the PM flag is set to "1", the register push/pop operation starts from the register number stored in RC[3:0].

∗2) <rs+1>, <rs+2>: contents of the registers that follow rs. (eg. rs=r0: <rs+1>=r1, <rs+2>=r2; rs=r15: <rs+1>=r0, <rs+2>=r1); They are incremented after each operation.

Extended instruction		Expansion format				
Opcode	Operand	Condition 1	Condition 2	Condition 3	Condition 4	Condition 5
xld.b **xld.ub** **xld.h** **xld.uh**	**%rd, [symbol+imm26]** **[symbol+imm26], %rs** *1	-medda32 not specified	–	–	–	–
	Example) xld.w %rd,[symbol+imm26]	ext (symbol+imm26) (25:13) ext (symbol+imm26) (12:0) ld.w %rd, [%r15]				
xld.w	**%rd, [%rb+imm26]** **[%rb+imm26], %rs** *1	imm26=0	1<imm26≤0x1fff	imm26>0x1fff	–	–
	Example) xld.w %rd,[%rb+imm26]	ld.w %rd,[%rb]	ext imm26(12:0) ld.w %rd,[%rb]	ext imm26(25:13) ext imm26(12:0) ld.w %rd,[%rb]		
xld.b **xld.ub**	**%rd, [%sp+imm32]** **[%sp+imm32], %rs** *1	imm32≤0x3f	0x3f<imm32≤0x7ffff	imm32>0x7ffff	–	–
	Example) xld.b %rd,[%sp+imm32]	ld.b %rd,[%sp+imm32(5:0)]	ext imm32(18:6) ld.b %rd,[%sp+imm32(5:0)]	ext imm32(31:19) ext imm32(18:6) ld.b %rd,[%sp+imm32(5:0)]		
xld.h **xld.uh**	**%rd, [%sp+imm32]** **[%sp+imm32], %rs** *1	imm32≤0x7f	0x7f<imm32≤0x7ffff	imm32>0x7ffff	–	–
	Example) xld.h %rd,[%sp+imm32]	ld.h %rd,[%sp+imm32(6:1)]	ext imm32(18:6) ld.h %rd,[%sp+imm32(5:0)]	ext imm32(31:19) ext imm32(18:6) ld.h %rd,[%sp+imm32(5:0)]		
xld.w	**%rd, [%sp+imm32]** **[%sp+imm32], %rs** *1	imm32≤0xff	0xff<imm32≤0x7ffff	imm32>0x7ffff	–	–
	Example) xld.w %rd,[%sp+imm32]	ld.w %rd,[%sp+imm32(7:2)]	ext imm32(18:6) ld.w %rd,[%sp+imm32(5:0)]	ext imm32(31:19) ext imm32(18:6) ld.w %rd,[%sp+imm32(5:0)]		
	%rd, sign32	32≤sign32≤31	-262144≤sign32<-32 or 31<sign32≤262143	sign32<-262144 or 262143<sign32	–	–
	Example) xld.w %rd,sign32	ld.w %rd,sign32(5:0)	ext sign32(18:6) ld.w %rd,sign32(5:0)	ext sign32(31:19) ext sign32(18:6) ld.w %rd,sign32(5:0)		
	%rd, symbol+imm32	Fixed	–	–	–	–
	Example) xld.w %rd,symbol+imm32	ext (symbol+imm32)@h ext (symbol+imm32)@m ld.w %rd,(symbol+imm32)@l				
	%rd, symbol-imm32	Fixed	–	–	–	–
	Example) xld.w %rd,symbol-imm32	ext (symbol-imm32)@h ext (symbol-imm32)@m ld.w %rd,(symbol-imm32)@l				

Remarks

*1) These operands are available only for xld.b, xld.h, xld.w instructions.

Expansion Format of Extended Instructions (2)

Extended instruction		Expansion format				
Opcode	Operand	Condition 1	Condition 2	Condition 3	Condition 4	Condition 5
xld.b **xld.ub** **xld.h** **xld.uh** **xld.w**	**%rd, [symbol+imm26]**	-medda32 specified	–	–	–	–
	Example) xld.w %rd,[symbol+imm26]	ext (symbol+imm26)@h ext (symbol+imm26)@m ld.w %rd,(symbol+imm26)@l ld.w %rd,[%rd]				
xld.b **xld.h** **xld.w**	**[symbol+imm26], %rs**	%rs≠%r0, -medda32 specified	%rs=%r0, -medda32 specified	–	–	–
	Example) xld.w [symbol+imm26],%rs	pushn %r0 ext (symbol+imm26)@h ext (symbol+imm26)@m ld.w %r0,(symbol+imm26)@l ld.w [%r0],%rs popn %r0	pushn %r1 (or push %r1)*1 ext (symbol+imm26)@h ext (symbol+imm26)@m ld.w %r1,(symbol+imm26)@l ld.w [%r1],%r0 popn %r1 (or pop %r1)*1			
xand **xoor** **xxor** **xnot**	**%rd, sign32**	-32≤sign32≤31	-262144≤sign32<-32 or 31<sign32≤262143	sign32<-262144 or 262143<sign32	–	–
	Example) xand %rd,sign32	and %rd,sign32(5:0)	ext sign32(18:6) and %rd,sign32(5:0)	ext sign32(31:19) ext sign32(18:6) and %rd,sign32(5:0)		
xadd **xsub**	**%rd, imm32**	imm32≤0x3f	0x3f<imm32≤0x7ffff	imm32>0x7ffff	–	–
	Example) xadd %rd,imm32	add %rd,imm32(5:0)	ext imm32(18:6) add %rd,imm32(5:0)	ext imm32(31:19) ext imm32(18:6) add %rd,imm32(5:0)		
xcmp	**%rd, sign32**	-32≤sign32≤31	262144≤sign32<-32 or 31<sign32≤262143	sign32<-262144 or 262143<sign32	–	–
	Example) xcmp %rd,sign32	cmp %rd,sign32(5:0)	ext sign32(18:6) cmp %rd,sign32(5:0)	ext sign32(31:19) ext sign32(18:6) cmp %rd,sign32(5:0)		
xsrl **xsll** **xsra** **xsla** **xrr** **xrl**	**%rd, imm5**	imm5≤8	8<imm5<16	imm5=16	16<imm5≤24	imm5>24
	Example) xsrl %rd,imm5	srl %rd,imm5(3:0)	srl %rd,0x8 srl %rd,imm5(2:0)	srl %rd,0x8 srl %rd,0x8	srl %rd,0x8 srl %rd,0x8 srl %rd,imm5(3:0)	srl %rd,0x8 srl %rd,0x8 srl %rd,0x8 srl %rd,imm5(2:0)

Remarks

*1) When the -mc33adv or -mc33pe option is specified.

Expansion Format of Extended Instructions (3)

Extended instruction		Expansion format				
Opcode	Operand	Condition 1	Condition 2	Condition 3	Condition 4	Condition 5
xbtst **xbclr** **xbset** **xbnot**	**[symbol+imm26], imm3**	-medda32 not specified	-medda32 specified	–	–	–
	Example) xbtst [symbol+imm26],imm3	ext (symbol+imm26)(25:13) ext (symbol+imm26)(12:0) btst [%r15],imm3	pushn %r0 ext (symbol+imm26)@h ext (symbol+imm26)@m ld.w %r0,(symbol+imm26)@l btst [%r0],imm3 popn %r0			
	[%rb+imm26], imm3	imm26=0	1<imm26≤0x1fff	imm26>0x1fff	–	–
	Example) xbtst [%rb+imm26],imm3	btst %rd,[%rb]	ext imm26(12:0) btst %rd,[%rb]	ext imm26(25:13) ext imm26(12:0) btst %rd,[%rb]		
scall **scall.d** **sjp** **sjp.d** **sjr*1** **sjr*1.d**	**label+imm32**	Fixed	–	–	–	–
	Example) scall label+imm32	ext (label+imm32)@rm call (label+imm32)@rl				
	sign22	-256≤sign22≤254	-2097152≤sign22<-256 or 254<sign22≤2097150	–	–	–
	Example) scall sign22	call sign22(8:1)	ext sign22(21:9) call sign22(8:1)			
xcall **xcall.d** **xjp** **xjp.d** **xjr*2** **xjr*2.d**	**label+imm32**	Fixed	–	–	–	–
	Example) xcall label+imm32	ext (label+imm32)@rh ext (label+imm32)@rm call (label+imm32)@rl				
	sign32	-256≤sign32≤254	-2097152≤sign32<-256 or 254<sign32≤2097150	sign32<-2097152 or 2097150<sign32	–	–
	Example) xcall sign32	call sign32(8:1)	ext sign32(21:9) call sign32(8:1)	ext sign32(31:22)<<0x3 ext sign32(21:9) call sign32(8:1)		

Remarks

*1) sjreq, sjreq.d, sjrne, sjrne.d, sjrgt, sjrgt.d, sjrge, sjrge.d, sjrlt, sjrlt.d, sjrle, sjrle.d, sjrugt, sjrugt.d, sjruge, sjruge.d, sjrult, sjrult.d, sjrule, sjrule.d

*2) xjreq, xjreq.d, xjrne, xjrne.d, xjrgt, xjrgt.d, xjrge, xjrge.d, xjrlt, xjrlt.d, xjrle, xjrle.d, xjrugt, xjrugt.d, xjruge, xjruge.d, xjrult, xjrult.d, xjrule, xjrule.d

Expansion Format of Extended Instructions (4) – C33 ADV Core

Extended instruction		Expansion format				
Opcode	Operand	Condition 1	Condition 2	Condition 3	Condition 4	Condition 5
ald.b **ald.ub** **ald.h** **ald.uh** **ald.w**	**%rd, [symbol+imm19]** **[symbol+imm19], %rs** ∗1 Example) ald.w %rd,[symbol+imm19]	-medda32 not specified	–	–	–	–
		ext (symbol+imm19) (18:6) ld.w %rd, [%dp+(symbol+imm19)(5:0)]				
xld.b **xld.ub** **xld.h** **xld.uh** **xld.w**	**%rd, [symbol+imm32]** **[symbol+imm32], %rs** ∗2 Example) xld.w %rd,[symbol+imm32]	-medda32 not specified	–	–	–	–
		ext (symbol+imm32) (31:19) ext (symbol+imm32) (18:6) ld.w %rd, [%dp+(symbol+imm32)(5:0)]				
xld.b **xld.ub**	**%rd, [%dp+imm32]** **[%dp+imm32], %rs** ∗2 Example) xld.b %rd,[%dp+imm32]	imm32≤0x3f	0x3f<imm32≤0x7ffff	imm32>0x7ffff	–	–
		ld.b %rd,[%dp+imm32(5:0)]	ext imm32(18:6) ld.b %rd,[%dp+imm32(5:0)]	ext imm32(31:19) ext imm32(18:6) ld.b %rd,[%dp+imm32(5:0)]		
xld.h **xld.uh**	**%rd, [%dp+imm32]** **[%dp+imm32], %rs** ∗2 Example) xld.h %rd,[%dp+imm32]	imm32≤0x7f	0x7f<imm32≤0x7ffff	imm32>0x7ffff	–	–
		ld.h %rd,[%dp+imm32(6:1)]	ext imm32(18:6) ld.h %rd,[%dp+imm32(5:0)]	ext imm32(31:19) ext imm32(18:6) ld.h %rd,[%dp+imm32(5:0)]		
xld.w	**%rd, [%dp+imm32]** **[%dp+imm32], %rs** ∗2 Example) xld.w %rd,[%dp+imm32]	imm32≤0xff	0xff<imm32≤0x7ffff	imm32>0x7ffff	–	–
		ld.w %rd,[%dp+imm32(7:2)]	ext imm32(18:6) ld.w %rd,[%dp+imm32(5:0)]	ext imm32(31:19) ext imm32(18:6) ld.w %rd,[%dp+imm32(5:0)]		
xsrl **xsll** **xsra** **xsla** **xrr** **xrl**	**%rd, imm5** Example) xsrl %rd,imm5	Fixed	–	–	–	–
		srl %rd,imm5				

Remarks

The extended instructions listed on this page are dedicated for the C33 ADV Core and can be used only when the -mc33adv option is specified.

∗1) These operands are available only for ald.b, ald.h, ald.w instructions.
∗2) These operands are available only for xld.b, xld.h, xld.w instructions.

Extended instruction		Expansion format				
Opcode	Operand	Condition 1	Condition 2	Condition 3	Condition 4	Condition 5
ald.b **ald.ub** **ald.h** **ald.uh** **ald.w**	**%rd, [symbol+imm19]** Example) ald.w %rd,[symbol+imm19]	-medda32 specified	–	–	–	–
		ext (symbol+imm19)@m ld.w %rd,(symbol+imm19)@l ld.w %rd,[%rd]				
ald.b **ald.h** **ald.w**	**[symbol+imm19], %rs** Example) ald.w [symbol+imm19],%rs	%rs≠%r0, -medda32 specified	%rs=%r0, -medda32 specified	–	–	–
		pushn %r0 ext (symbol+imm19)@m ld.w %r0,(symbol+imm19)@l ld.w [%r0],%rs popn %r0	push %r1 ext (symbol+imm19)@m ld.w %r1,(symbol+imm19)@l ld.w [%r1],%r0 pop %r1			
xld.b **xld.ub** **xld.h** **xld.uh** **xld.w**	**%rd, [symbol+imm32]** Example) xld.w %rd,[symbol+imm32]	-medda32 specified	–	–	–	–
		ext (symbol+imm32)@h ext (symbol+imm32)@m ld.w %rd,(symbol+imm32)@l ld.w %rd,[%rd]				
xld.b **xld.h** **xld.w**	**[symbol+imm32], %rs** Example) xld.w [symbol+imm32],%rs	%rs≠%r0, -medda32 specified	%rs=%r0, -medda32 specified	–	–	–
		pushn %r0 ext (symbol+imm32)@h ext (symbol+imm32)@m ld.w %r0,(symbol+imm32)@l ld.w [%r0],%rs popn %r0	push %r1 ext (symbol+imm32)@h ext (symbol+imm32)@m ld.w %r1,(symbol+imm32)@l ld.w [%r1],%r0 pop %r1			

Remarks

The extended instructions listed on this page are dedicated for the C33 ADV Core and can be used only when the -mc33adv option is specified.

EPSON

International Sales Operations

AMERICA

EPSON ELECTRONICS AMERICA, INC.

2580 Orchard Parkway,
San Jose, CA 95131, USA
Phone: +1-800-228-3964 FAX: +1-408-922-0238

EUROPE

EPSON EUROPE ELECTRONICS GmbH

Riesstrasse 15, 80992 Munich,
GERMANY
Phone: +49-89-14005-0 FAX: +49-89-14005-110

ASIA

EPSON (CHINA) CO., LTD.

7F, Jinbao Bldg., No.89 Jinbao St.,
Dongcheng District,
Beijing 100005, CHINA
Phone: +86-10-6410-6655 FAX: +86-10-6410-7320

SHANGHAI BRANCH

7F, Block B, Hi-Tech Bldg., 900 Yishan Road,
Shanghai 200233, CHINA
Phone: +86-21-5423-5522 FAX: +86-21-5423-5512

SHENZHEN BRANCH

12/F, Dawning Mansion, Keji South 12th Road,
Hi-Tech Park, Shenzhen 518057, CHINA
Phone: +86-755-2699-3828 FAX: +86-755-2699-3838

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 TAIWAN TECHNOLOGY & TRADING LTD.

14F, No. 7, Song Ren Road,
Taipei 110, TAIWAN
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 CORP.
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

SEIKO EPSON CORP.
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

S5U1C33001C Manual
(C/C++ Compiler Package for S1C33 Family) (Ver. 3.3.0)

SEIKO EPSON CORPORATION
SEMICONDUCTOR OPERATIONS DIVISION

■ **EPSON Electronic Devices Website**

http://www.epson.jp/device/semicon_e

Document code: 410561105
First Issue May, 2006 ⓁB
Revised February 2009, Japan Ⓓ