EPSON

CMOS 16-BIT SINGLE CHIP MICROCONTROLLER
(S1C17 Family C Compiler Package Ver. 3.2)

GNU17 Ver. 3.2
Tutorial

SEIKO EPSON CORPORATION

Rev. 1.1

NOTICE: PLEASE READ THE FOLLOWING NOTICE CAREFULLY BEFORE USING THIS DOCUMENT

The contents of this document are subject to change without notice.

1. This document may not be copied, reproduced, or used for any other purpose, in whole or in part, without the consent of the Seiko Epson Corporation (“Epson”).
2. Before purchasing or using Epson products, please contact our sales representative for the latest information and always be sure to check the latest information published on Epson’s official web sites and other sources.
3. Information provided in this document such as application circuits, programs, usage, etc., are for reference purposes only. Using the application circuits, programs, usage, etc. in the design of your equipment or systems is your own responsibility. Epson makes no guarantees against any infringements or damages to any third parties’ intellectual property rights or any other rights resulting from the information. This document does not grant you any licenses, intellectual property rights or any other rights with respect to Epson products owned by Epson or any third parties.
4. Epson is committed to constantly improving quality and reliability, but semiconductor products in general are subject to malfunction and failure. By using Epson products, you shall be responsible for your hardware. Software and systems must be designed well enough to prevent death or injury as well as any property damage even if any of the malfunctions or failures might be caused by Epson products. When designing your products using Epson products, please be sure to check and comply with the latest information regarding Epson products (this document, specifications, data sheets, manuals, Epson’s web site, etc.). When using the information included above materials such as product data, charts, technical contents, programs, algorithms and application circuit examples, you shall evaluate your products both on a stand-alone basis as well as within your overall systems. You shall be solely responsible for deciding whether or not to adopt and use Epson products.
5. Epson has prepared this document and programs provided in this document carefully to be accurate and dependable, but Epson does not guarantee that the information and the programs are always accurate and complete. Epson assumes no responsibility for any damages which you incur due to misinformation in this document and the programs.
6. No dismantling, analysis, reverse engineering, modification, alteration, adaptation, reproduction, etc., of Epson products is allowed.
7. Epson products have been designed, developed and manufactured to be used in general electronic applications (office equipment, communications equipment, measuring instruments, home electronics, etc.) (“General Purpose”) and applications which is individually listed in this document or designated by Epson (“Designated Purpose”). Epson products are NOT intended for any use beyond the General Purpose and Designated Purpose uses that requires particular/higher quality or reliability in order to refrain from causing any malfunction or failure leading to death, injury, serious property damage or severe impact on society, including, but not limited to those listed below (“Particular Purpose”). Therefore, you are advised to use Epson products only for General Purpose and Designated Purpose uses. Should you desire to buy and use Epson products for a Particular Purpose, Epson makes no warranty and disclaims with respect to Epson products, whether express or implied, including without limitation any implied warranty of merchantability or fitness for any Particular Purpose. Please be sure to contact our sales representative and obtain approval in advance.

   [Examples of Particular Purpose]
   Space equipment (artificial satellites, rockets, etc.) /
   Transportation vehicles and their control equipment (automobiles, aircraft, trains, ships, etc.) /
   Medical equipment / Relay equipment to be placed on ocean floor /
   Power station control equipment / Disaster or crime prevention equipment / Traffic control equipment / Financial equipment

   Other applications requiring similar levels of reliability as those listed above. Please be sure to contact our sales representative for details of the other applications.
8. Epson products listed in this document and our associated technologies shall not be used in any equipment or systems that laws and regulations in Japan or any other countries prohibit to manufacture, use or sell. Furthermore, Epson products and our associated technologies shall not be used for developing weapons of mass destruction, or any other military purposes or applications. If exporting Epson products or our associated technologies, you shall comply with the Foreign Exchange and Foreign Trade Control Act in Japan, Export Administration Regulations in the U.S.A. (EAR) and other export-related laws and regulations in Japan and any other countries and follow the required procedures as provided by the relevant laws and regulations.
9. Epson assumes no responsibility for any damages (whether direct or indirect) caused by or in relation with your non-compliance with the terms and conditions in this document.
10. Epson assumes no responsibility for any damages (whether direct or indirect) incurred by any third party that you assign, transfer, loan, etc., Epson products to.
11. For more details or other concerns about this document, please contact our sales representative.
12. Company names and product names listed in this document are trademarks or registered trademarks of their respective companies.

Evaluation board/kit and Development tool important notice

1. Epson evaluation board/kit or development tool is designed for use for engineering evaluation, demonstration, or development purposes only. Do not use it for other purposes. It is not intended to meet the requirements of design for finished products.
2. Epson evaluation board/kit or development tool is intended for use by an electronic engineer and is not a consumer product. The user should use it properly and in a safe manner. Epson dose not assume any responsibility or liability of any kind of damage and/or fire coursed by the use of it. The user should cease to use it when any abnormal issue occurs even during proper and safe use.
3. The part used for Epson evaluation board/kit or development tool may be changed without any notice.

Rev. e1.4, 2023. 4

Preface

This tutorial is intended to help designers and programmers, who develop a product using an S1C17 Family microcontroller, to understand the embedded program development procedure using the S1C17 Family software development tool, "S1C17 Family C Compiler Package."

Reference documents

For the contents not described in this tutorial, please refer to the manuals and documents shown below as necessary.

Contents to be referred	Reference documents
C language (ANSI C compliant) and C source coding method	General books that describe ANSI C
Basic operating methods for GNU17 IDE (Eclipse IDE for C/C++ Developers Package)	General books that describe Eclipse IDE for C/C++ Developers Package
GNU C, binutils, and the linker script for the GNU linker (ld)	GNU tool manuals
Basic operating methods for Windows	Windows manuals
S1C17 Family instruction set	S1C17 Family S1C17 Core Manual
S1C17 Family development tools	S5U1C17001C Manual and hardware development tool manuals
S1C17 Family microcontroller	S1C17xxx Technical Manual

Operating environment, installation method

For the operating environment, installation method, and the folder configuration after being installed, refer to the Readme file included in the package.

Localization

The GNU17 IDE is developed based on the Eclipse IDE for C/C++ Developers Package and the user interface of the workbench installed uses English for its display. If you need to localize it, install a language pack. As an example, Appendix B shows a method to install a language pack of the Babel project from the GNU17 IDE menu.

This tutorial uses the original user interface (in English) without being localized.

– Contents –

1 Software Development Flow

The S1C17 Family C Compiler Package (S5U1C17001C) provides an integrated development environment (GNU17 IDE) for developing software to be embedded in the S1C17 Family microcontrollers. The GNU17 IDE is capable of being used for creating source files, compiling/assembling, linking, debugging, and generating the final submission data file. This tutorial describes basic software development flows with the GNU17 IDE operation procedures.

1.1 Configuration of Software Development Tools

Figure 1.1.1 shows the configuration of the software development tools included in the S1C17 Family C Compiler Package (S5U1C17001C).

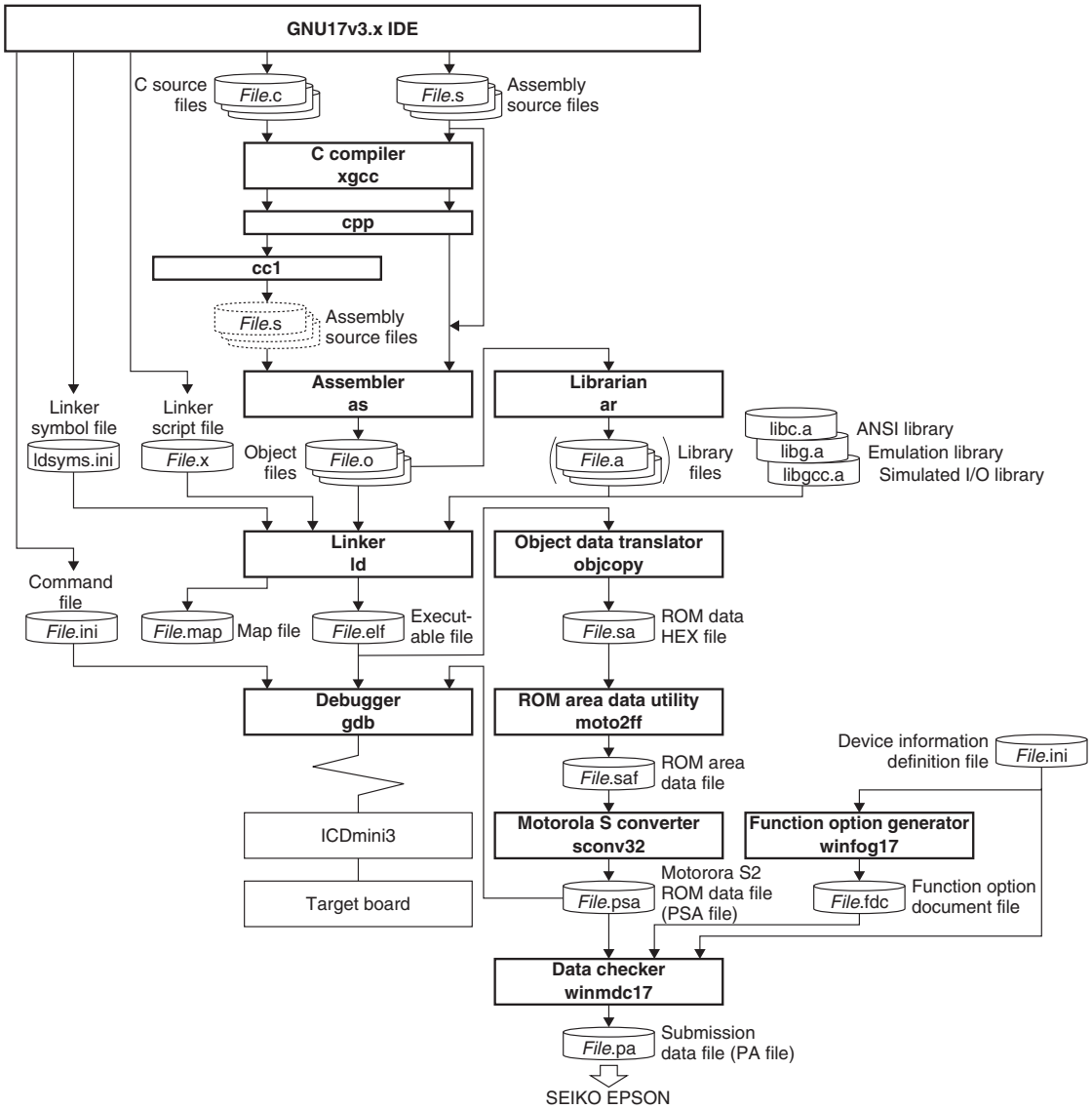


Figure 1.1.1 Configuration of Software Development Tools and Files

1.2 Software Development Using GNU17 IDE

This section gives an overview of the software development procedure using GNU17 IDE (hereinafter referred to as IDE). The actual operation procedures are detailed in the tutorial chapters.
Figure 1.2.1 shows a basic software development procedure using the IDE.

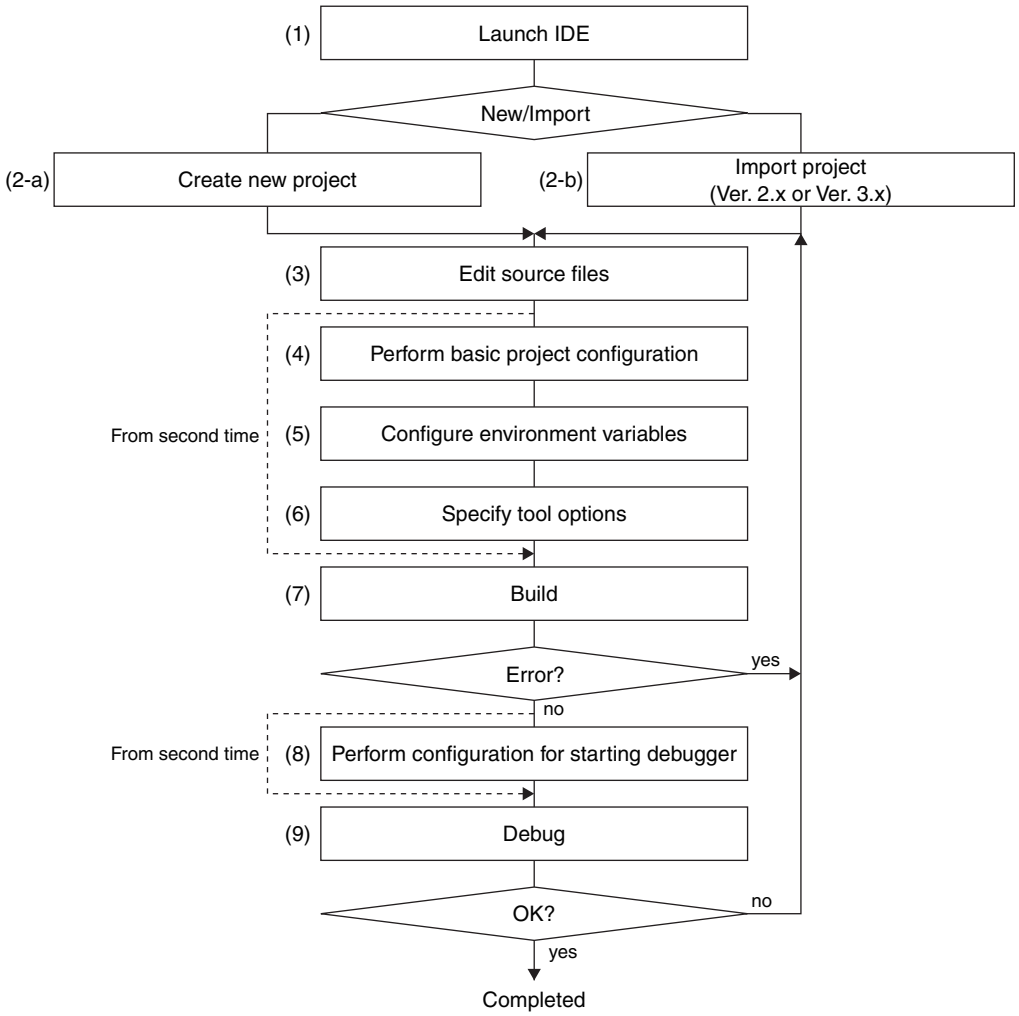


Figure 1.2.1 Software Development Procedure using IDE

(1) Launching the IDE

The IDE is an integrated development environment software, which is an Eclipse with an S1C17 Family plugin added. This workbench provides functions for creating source files, building projects, debugging programs, and generating the PA file to be submitted to Seiko Epson.
First launch the IDE on Windows. The processing and operations shown below can be performed in the IDE.

(2) Creating a new project/importing a project

In the software development using the IDE, a project folder is created for each application to be developed to manage the software resources required for the application.

(2-a) Creating a new project

If there is no available project, create a new project.

(2-b) Importing a project

If an available project has already been created using the IDE, it is possible to migrate the project from another environment or to upgrade the program by importing the project folder.

Note: GNU17 Ver. 3.x has a newly added library (startup processing library) for processing the interrupt vector table and boot sequence. Basically, the IDE uses this library to build projects. To support the projects created by GNU17 Ver. 2.x that does not have this library file, the IDE has a function to import Ver. 2.x projects as well as the current version (GNU17 Ver. 3.x) project import function. For more information, refer to the tutorials described later.

(3) Editing the source program

Add the source files to the project.

Source files can be created and edited using the editor of the IDE. Existing source files including those that are created using a general-purpose editor can also be imported and edited.

Note: When a GNU17 Ver. 2.x project is imported, the source file(s) must be edited to adopt to the new functions of Ver. 3.x. For more information, refer to the tutorials described later.

(4) Performing basic project configuration

Set the program type (executable file/library file), target CPU, memory model (16MB/64KB), and GCC version. Most of steps (5) and (6) below are automatically configured from these settings.

(5) Configuring environment variables

Specify the user library to be linked and the coprocessor type of the target CPU as necessary.

(6) Specifying tool options

Specify the C compiler, assembler, and linker startup options, and specify/edit the linker script file to be used for linkage, as necessary.

Steps (4) to (6) set the information required for building the project. Once they have been set, these steps can be skipped from the second time.

(7) Building the project

Execute the build process after the source program has been created/edited and the configuration for building has been finished. The processing shown below is sequentially executed, and an elf format object file that can be used for debugging and a submission data file (PA file) are generated.

Compile (for C sources)

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

Assemble (for assembler sources)

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

If the source files contain preprocessor instructions, use "xgcc" to perform preprocessing and assembly. When the necessary options are specified, "xgcc" executes the preprocessor "cpp" and the assembler "as."

Link

One or more object files that are generated by compiling and assembling are combined into one elf format object file that includes the information required for debugging and is executable by arranging in the ROM.

Converting into S-record format

The IDE launches objcopy, moto2ff, and sconv32 sequentially to extract the ROM data from the elf format object file and generates a Motorola S2 format PSA file in which the unused area is filled with 0xff.

Use the PSA file to perform final verification of program operation on the actual target board.

Generating the submission data file (PA file)

The IDE launches winmdc17 to generate the PA file to be submitted to Seiko Epson from the PSA file. This file is necessary when you request Seiko Epson to program the ROM or flash memory embedded in the MCU with the user program. The submission PA file must be generated from the PSA file after the operation check has been completed.

If the MCU has a function option, winfog17 and winmdc17 must be launched separately from the build process to generate the PA file. For more information, refer to the "Technical Manual" for the MCU and the "S5U1C17001C Manual."

(8) Performing debugger startup configuration

Specify the command file used for starting up the debugger and edit the commands written to the file as necessary. This configures the information required for starting the debugger. Once this operation has been done, this step can be skipped from the second time.

(9) Debugging

Launch the debugger "gdb" and perform an operation check and debugging of the program using the elf format object file generated by the linker "ld" or the S-record format PSA file. By using ICDmini, the hardware operation can be debugged. Also the debugger "gdb" has simulator mode to simulate the S1C17 Family microcontroller operations on the PC.

(10) Generating a library file

In addition to the tools described above, the librarian "ar" is provided. This tool organizes modules for general-purpose processing (e.g., object files output by the assembler "as") as a library, facilitating future applications development involving the S1C17 Family microcontrollers.

2 Tutorial 1
(Basic Operations from Project Creation to Debug)

This tutorial gives you a quick tour of a basic operation procedure from startup of the IDE to debugging the program. For detailed information on each tool, refer to the "S5U1C17001C Manual."

Files used

This tutorial assumes that the sample source files listed below exist in the "C:\EPSON\GNU17V3\sample\tutorial" directory.

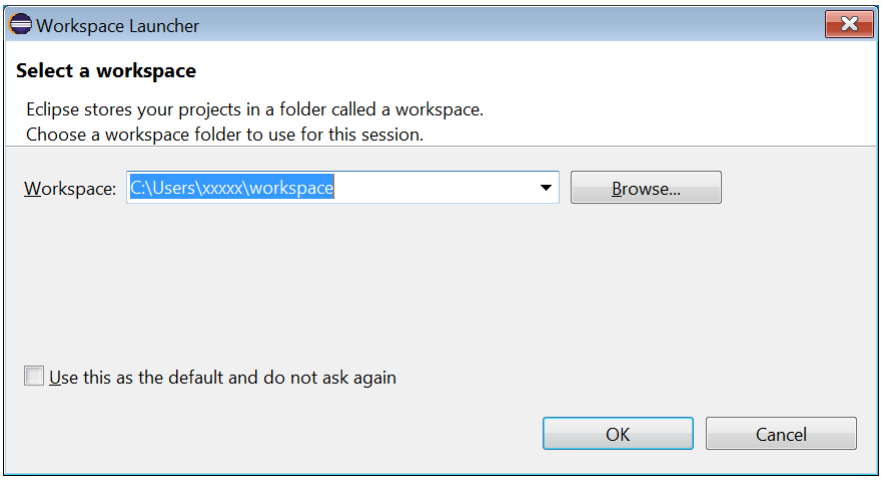
\src\tutorial.c
\src\sub.c
\inc\sub.h

The following shows an operation procedure for creating a project, building a program, and debugging the program using the source files above. Depending on the use environment, the display examples may be different from those that are displayed on your PC.

2.1 Launching the IDE

Step 1: Launch the IDE by selecting [EPSON MCU] > [GNU17V3] > [GNU17V3 IDE] from the Windows Start menu (or double-click the eclipse.exe icon [] that exists in the "C:\EPSON\GNU17V3\eclipse" directory).

After an Eclipse splash screen is displayed, the [Workspace Launcher] dialog box shown below appears. Specify the workspace (directory) where the project resources and output files will be stored.

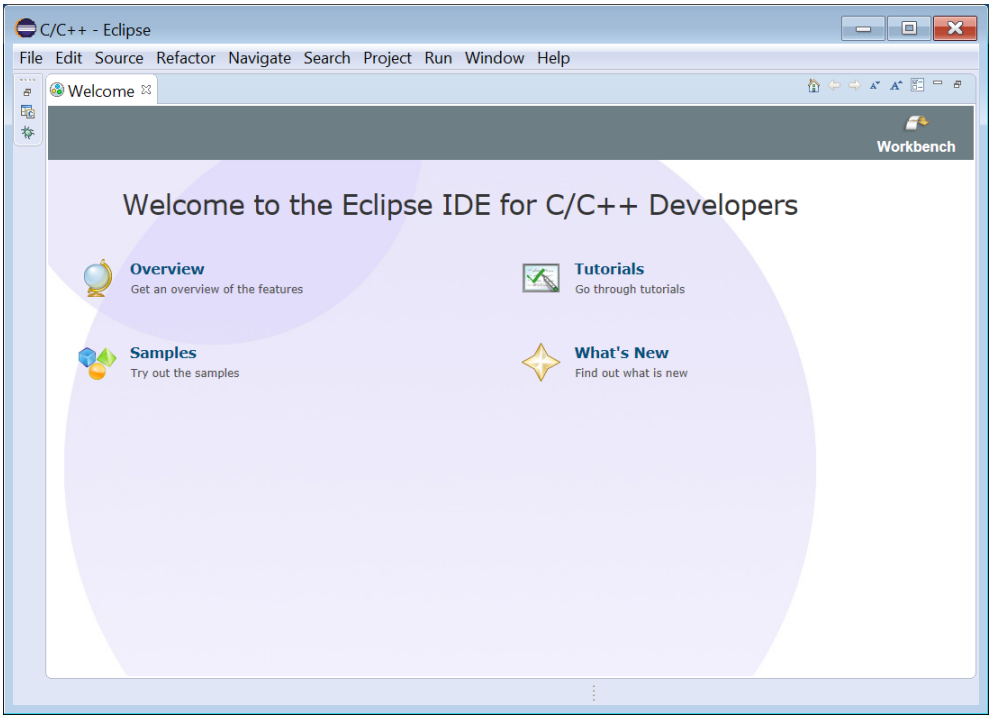


Although any directory can be selected or a new directory can be created as the workspace, this tutorial uses the default workspace directory.

Note: 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.

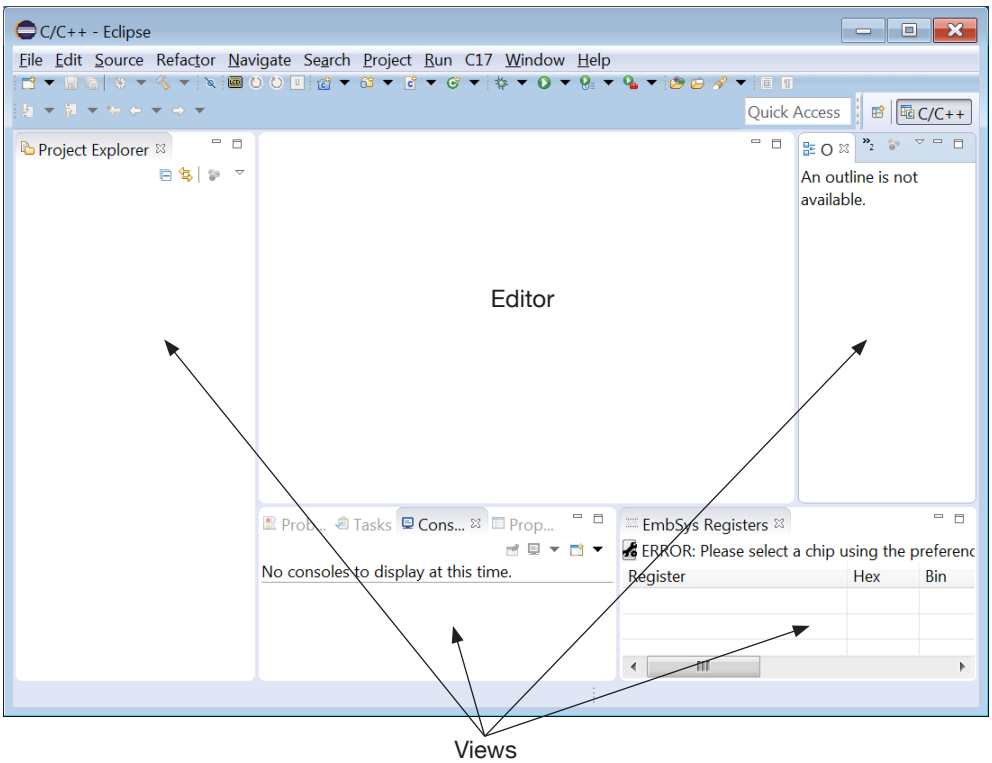
The IDE window shown below will open.



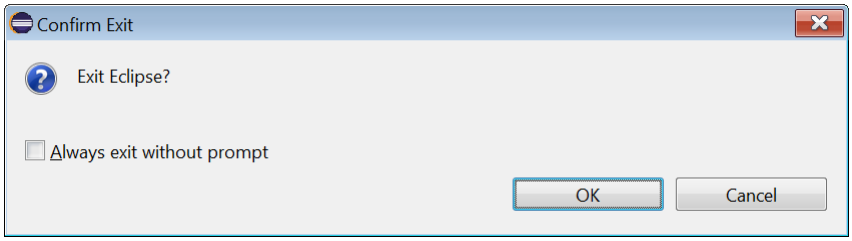
The [Welcome] page is displayed at initial start-up of the IDE.

Step 3: Click the ⊠ (Close) button on the [Welcome] tab to close the [Welcome] page.

* The [Welcome] page will not be displayed at subsequent IDE start-ups. To display it again, select [Welcome] from the [Help] menu.



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



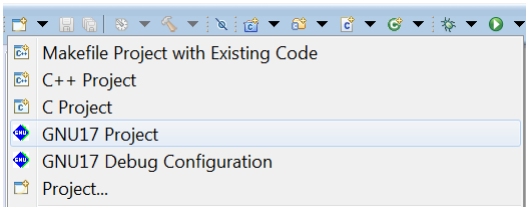
2.2 Creating a Project

In an application development, a single executable program file is created from multiple files such as source files and header files. To collectively manage these files, a project should be created. The IDE generates a folder with a specified project name for each project.

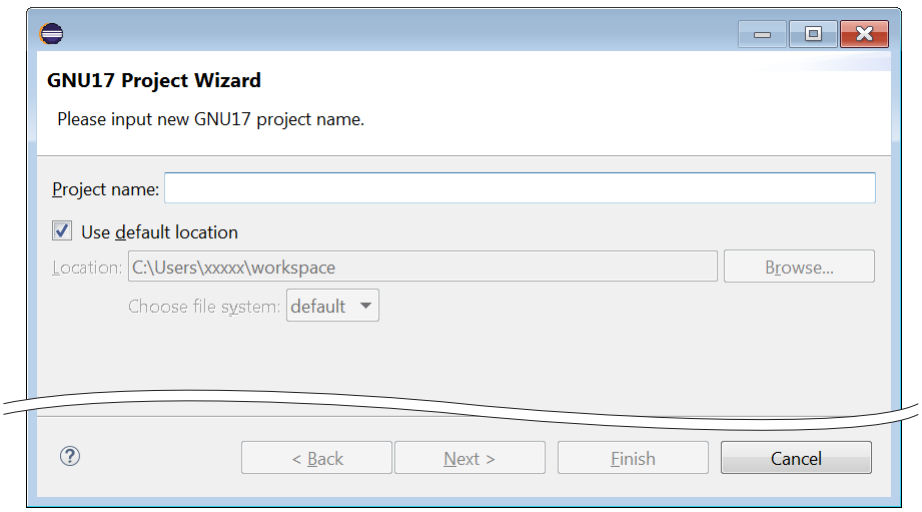
To create a new project

Step 1: Select [New GNU17 Project] from the (New) drop-down list* in the toolbar.

* This can also be selected from the [File] menu > [New] and the (New C/C++ Project) drop-down list in the toolbar.

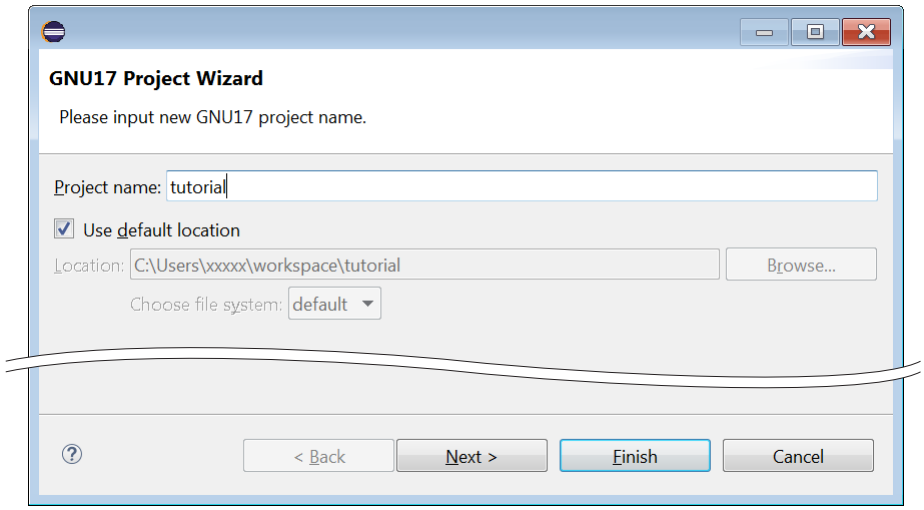


The [New GNU17 Project] wizard will start.



Specifying the project name

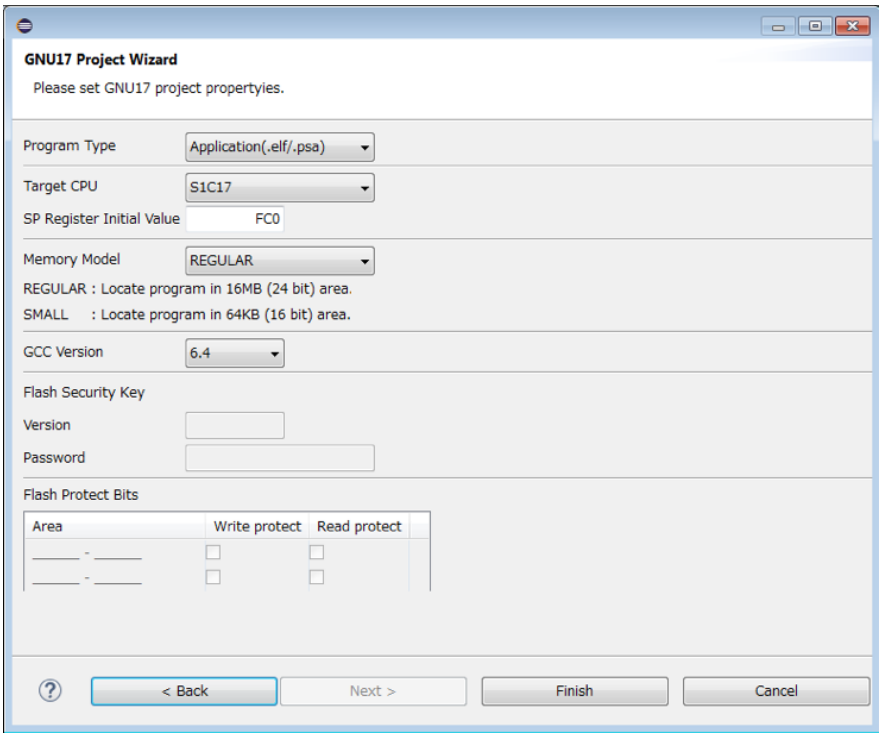
Step 2: Enter the project name “tutorial” at [Project name:].



By selecting the [Use default location] check box, a project folder named “tutorial” is generated in the workspace directory specified at start of the IDE.
The name specified here will also be assigned to the executable object file (.elf/.psa) to be generated by building the project.

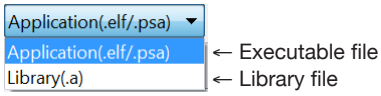
Step 3: Click the [Next >] button.

The wizard goes to the next page to configure the project properties.



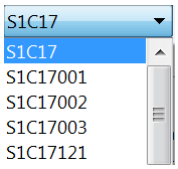
Selecting a program type

Step 4: Select the program type to be generated from the [Program Type] drop-down list. Select [Application (.elf/.psa)] here.



Specifying a target CPU

Step 5: Select the S1C17 MCU to be used for the application from the [Target CPU] drop-down list. Select [S1C17] (model independent) here.

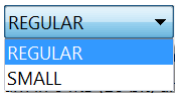


Setting the SP register initial value

Step 6: Set the SP register (stack pointer) initial value to [SP Register Initial Value]. Normally it is set to an appropriate value according to the RAM capacity of the selected target CPU as displayed here, so changing is not necessary.

Selecting a memory model

Step 7: Select the memory model supported by the S1C17 MCU to be used for the application from the [Memory Model] drop-down list. Select [REGULAR] here.



Selecting a GCC version

Step 8: Select the version of the C compiler “gcc” to be used from the [GCC Version] drop-down list. Normally, leave this as [4.9].

Setting the flash security key

Step 9: If the target CPU supports the flash security function and the flash memory has been protected with a password stored, enter the password at [Password] for canceling the flash protection. It is not necessary to set in this tutorial.

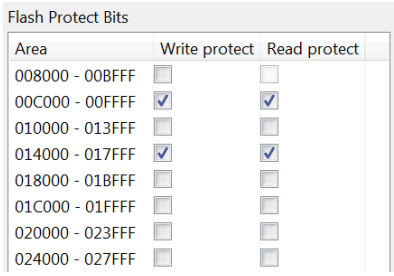
The password given here is used when creating the submission data file (PA file) using winmdc17. It is also used as an argument of the “c17 pwul” command in the debugger startup command file.
The version is automatically entered to [Version] according to the target CPU selected. If the selected target CPU does not support the flash security function, the [Version] and [Password] text boxes will be disabled.

Setting the flash protect bits

Step 10: In the [Flash Protect Bits] field, select the flash areas to be protected. This operation is not needed in this tutorial.

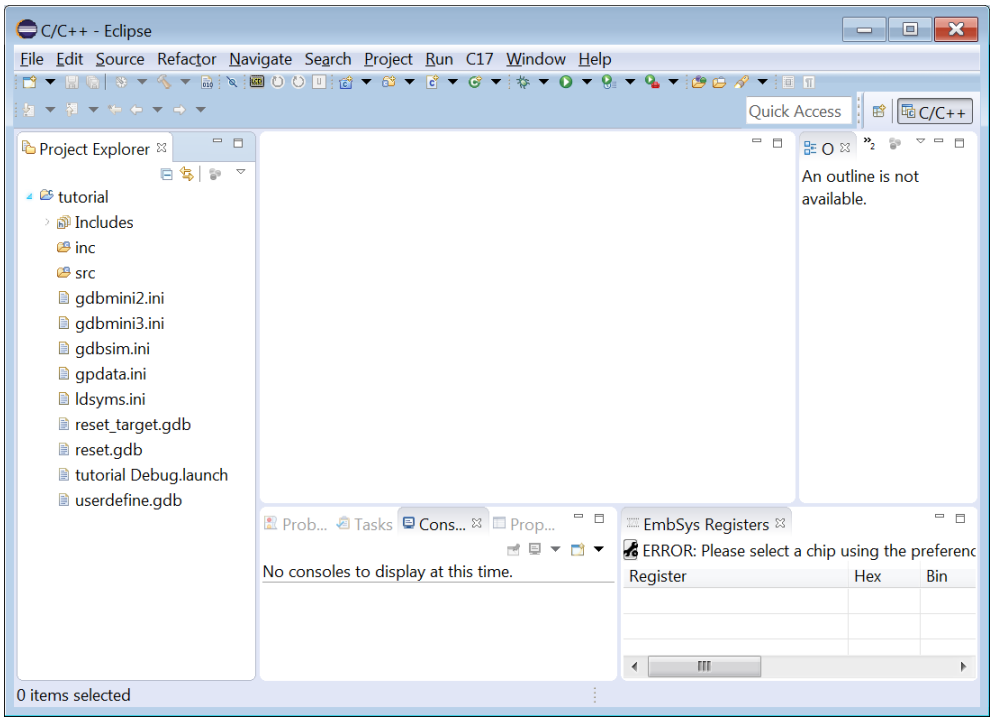
When a target CPU that supports protect bits is selected, the [Area] field displays the start and end addresses of the protective areas.
Selecting the [Write protect] or [Read Protect] check box on the right enables write protection or read protection (or both) so that the area is prohibited from writing or reading data. To configure an area to be enabled to write and/or read, deselect the check box. For the flash areas to be debugged, deselect both of the check boxes.

Example: Address ranges from 0xc000 to 0xffff and from 0x14000 to 0x17fff are configured as a read/write protect area.



When read/write protection is enabled, the IDE will generate a flash-protected psa file with the name "*<project name>*_ptd.psa".

Step 11: Click the [Finish] button.



The IDE exits the [GNU17 Project] wizard after generating a project with the specified name.
The target CPU, SP register initial value, memory model, GCC version, flash security key, and flash protect bits settings can be changed later.

Folders/files generated

The following folders and files are created in the "tutorial" project folder generated.

tutorial Project folder
Includes Include paths
C:/EPSON/GNU17V3/gcc6/include Path to gcc ver. 6.4 header files
tutorial/inc Path to tutorial project header files

inc	Folder for storing tutorial project header files
src	Folder for storing tutorial project source files
gdbmini2.ini	Debugger startup command file (for ICDmini (ICDmini2) mode)
gdbmini3.ini	Debugger startup command file (for ICDmini (ICDmini3) mode)
gdbsim.ini	Debugger startup command file (for simulator mode)
gpdata.ini	gpdata option file (for Gang Programmer)
ldsyms.ini	Linker symbol file
reset_target.gdb	GDB command file (for target reset)
reset.gdb	GDB command file (for CPU reset)
tutorial Debug.launch	Debugger startup configuration file
userdefine.gdb	GDB command file (for user definition)

2.3 Creating and Importing Source Files

The IDE supports source files created in C and assembler languages to generate an object. All source files required to generate an object must be imported into the project created earlier.

2.3.1 Creating Source Files

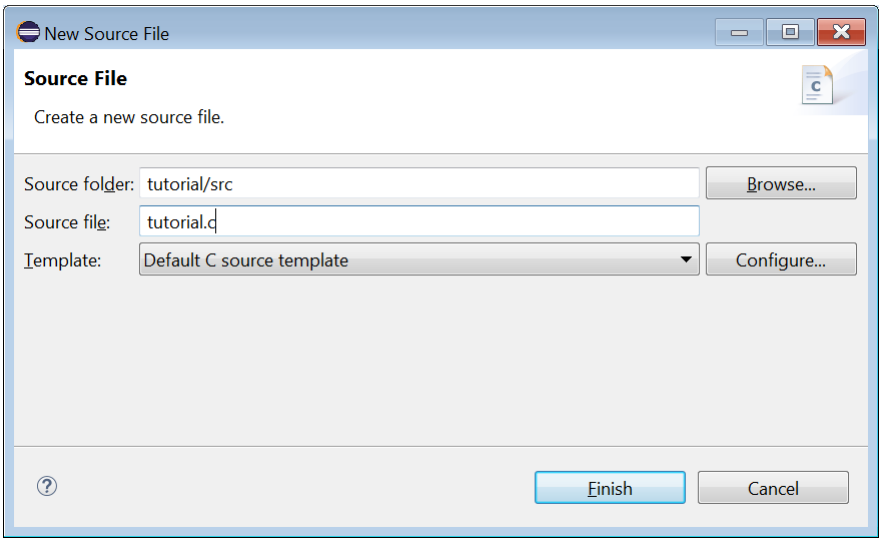
Create source files using the IDE editor or a general-purpose editor. Existing source files for S1C17 Family applications can also be used.
Sample source files are provided for this tutorial, so it is not necessary to create a new source file. The Steps 1 and 2 below show a new creation procedure for reference (it is not necessary to operate here).

To create a new source/header file

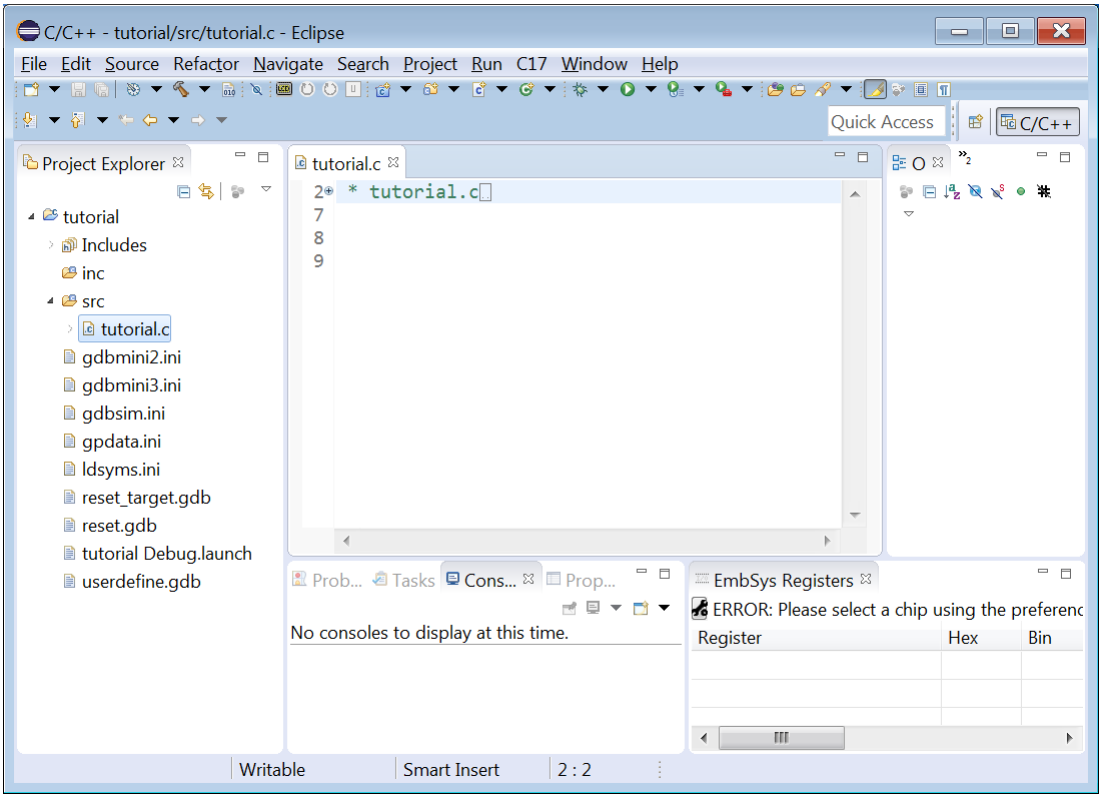
Step 1: Select “tutorial” > “src” in the [Project Explorer] view and select [Source File] from the (New) drop-down list* in the tool bar. (Select “tutorial” > “inc” and select [New] > [Header File] when creating a header file.)

* This can also be selected from the [File] menu > [New] and the (New C/C++ Source File) drop-down list in the toolbar.

This will bring up the [New Source File] dialog box.



Step 2: Enter the source file name (header file name) to be created to [Source file:] ([Header file:]) and then click the [Finish] button.



A file with the specified name will be created in the folder selected or specified. The name of the file created appears in an editor tab and the editor is ready to start entering text.

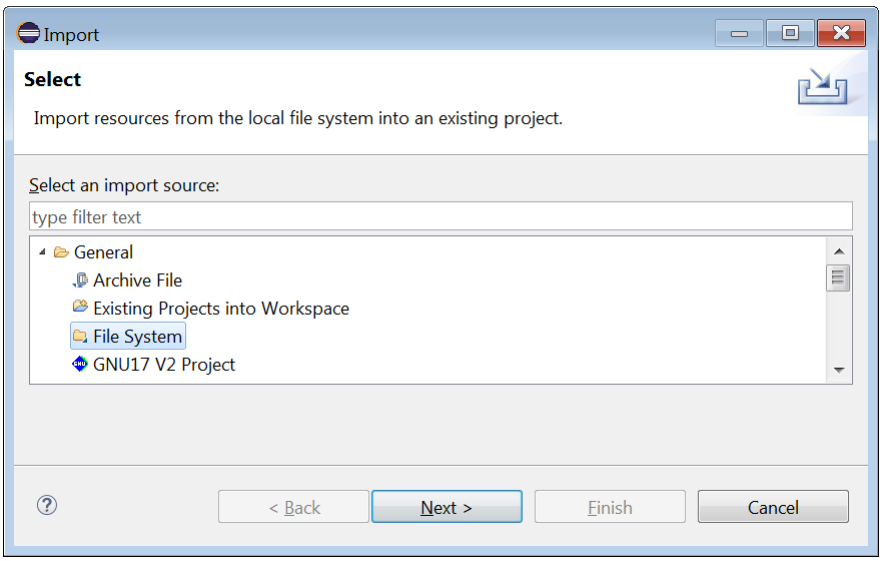
2.3.2 Importing Source Files

Import the sample source files into the project.

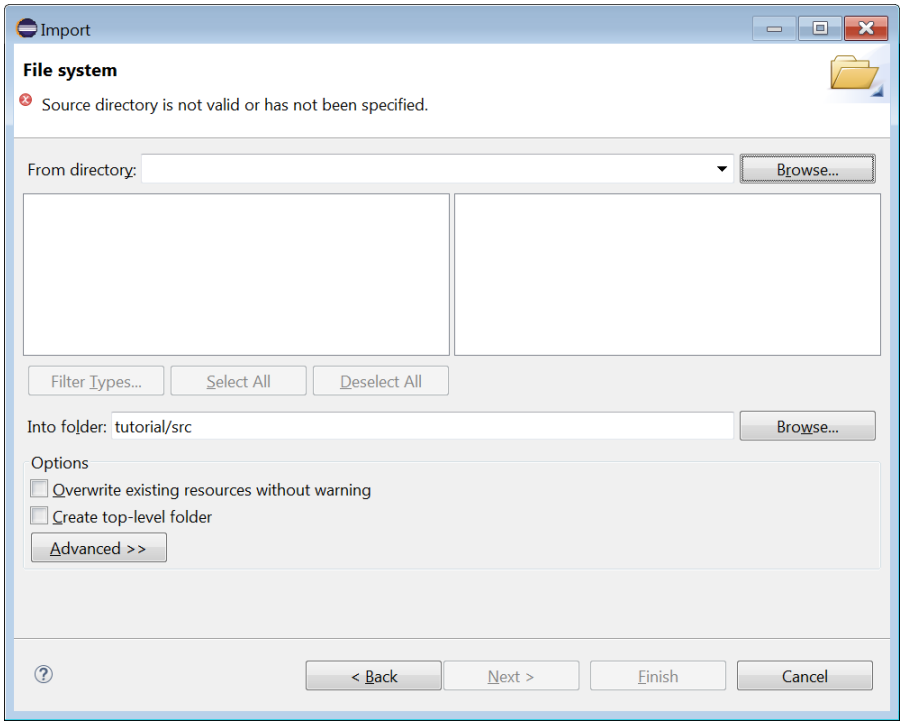
To import a source/header file

Step 3: Select “tutorial” > “src” in the [Project Explorer] view and select [Import...] from the [File] menu.

The [Import] wizard will start.

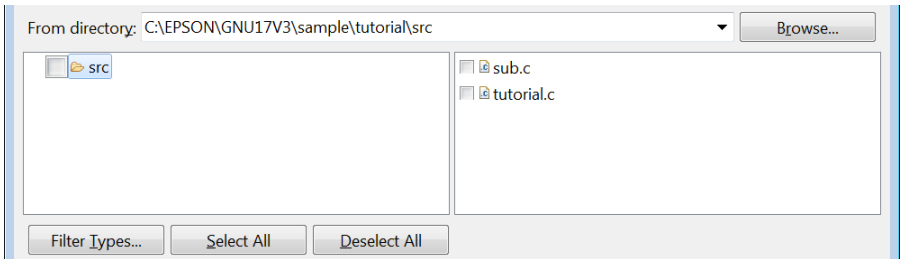


Step 4: Select [General] > [File System] from the list displayed and then click the [Next >] button.

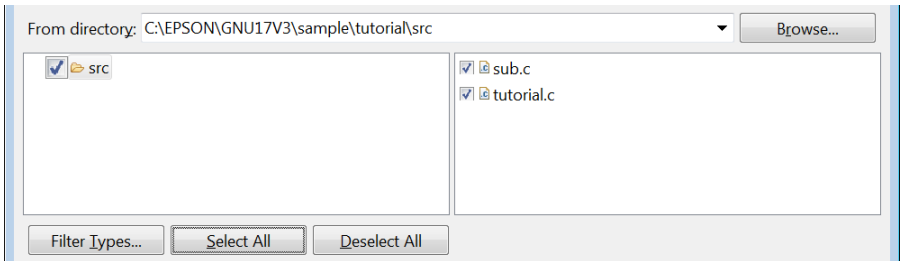


Step 5: Click the [Browse...] at [From directory:] to bring up the [Import from directory] dialog box. Select the "\EPSON\GNU17V3\sample\tutorial\src" directory in the drive (C) in which the IDE is installed and click the [OK] button.

The directory selected is shown in the left list box and the list of files that exist in the directory is shown in the right list box.



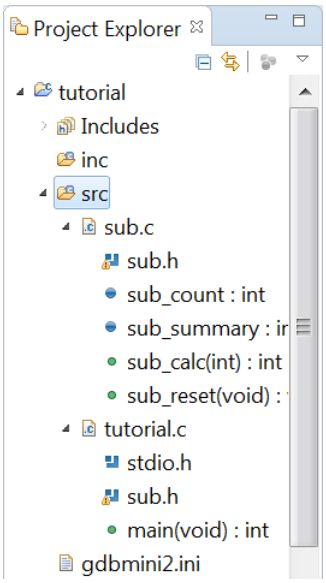
Step 6: Click the [Select All] button to select the source files.



Step 7: Check to see if the contents above are displayed and then click the [Finish] button.

The "sub.c" and "tutorial.c" source files are imported into the project with the operations above.

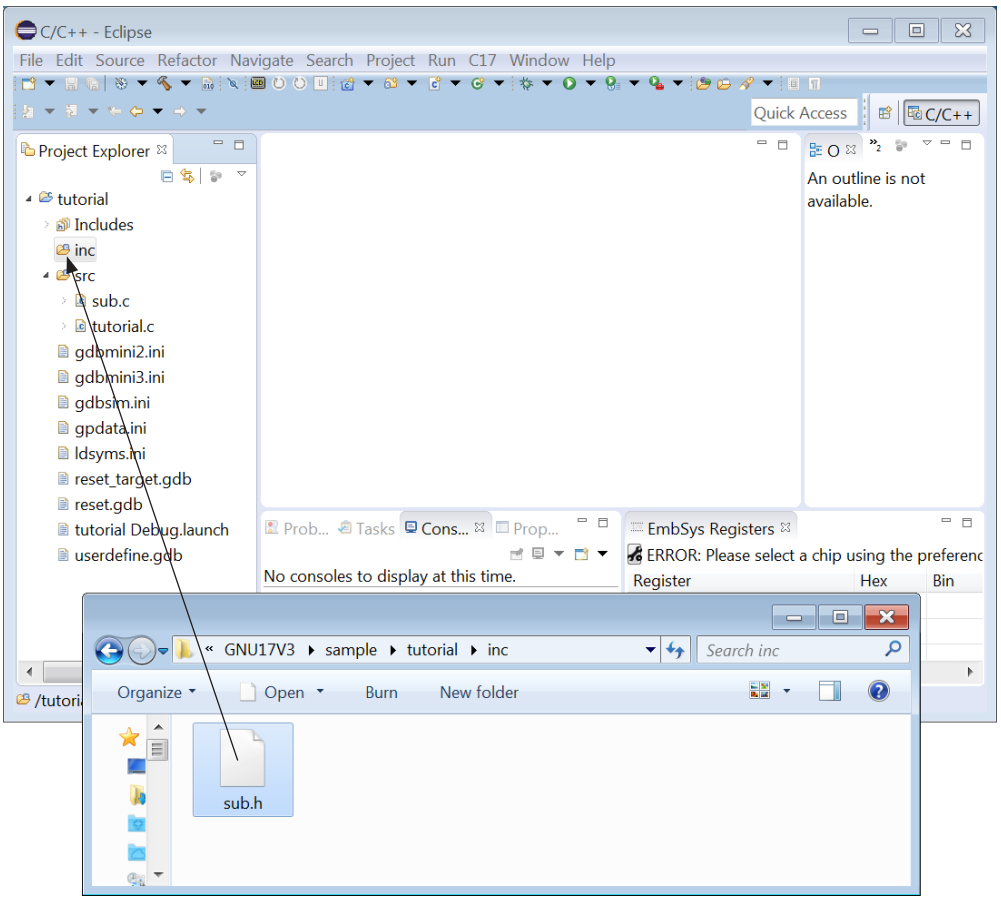
Step 8: Expand “tutorial” > “src” > “sub.c/tutorial.c” in the [Project Explorer] view.



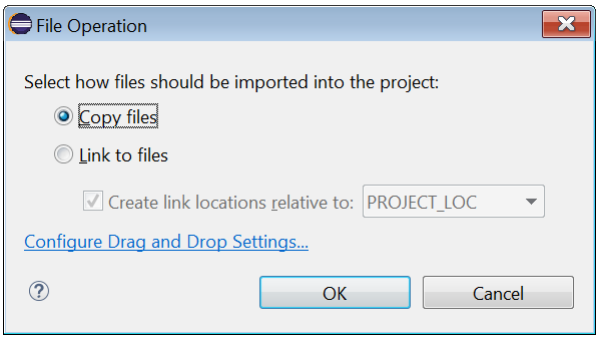
The source files added to the “tutorial\src” directory in the [Project Explorer] view, the header files included in them, and the global variables and functions defined in them are displayed.

Step 9: Import the header file “C:\EPSON\GNU17V3\sample\tutorial\inc\sub.h.”

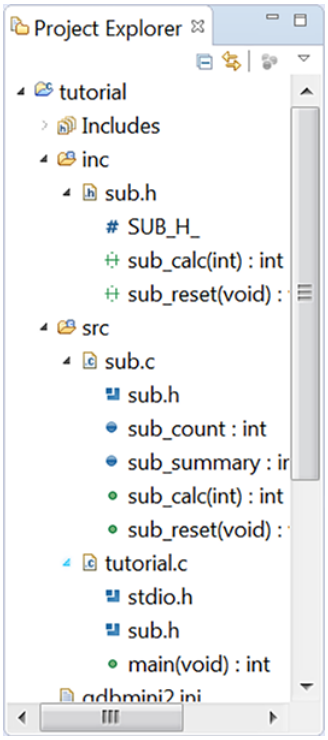
The file can be imported by dragging and dropping on the [Project Explorer] view from Windows Explorer. (Importing into the “inc” folder in the “tutorial” project)



Dragging and dropping a file will bring up the [File Operation] dialog box.



Step 10: Select [Copy files] and click the [OK] button.

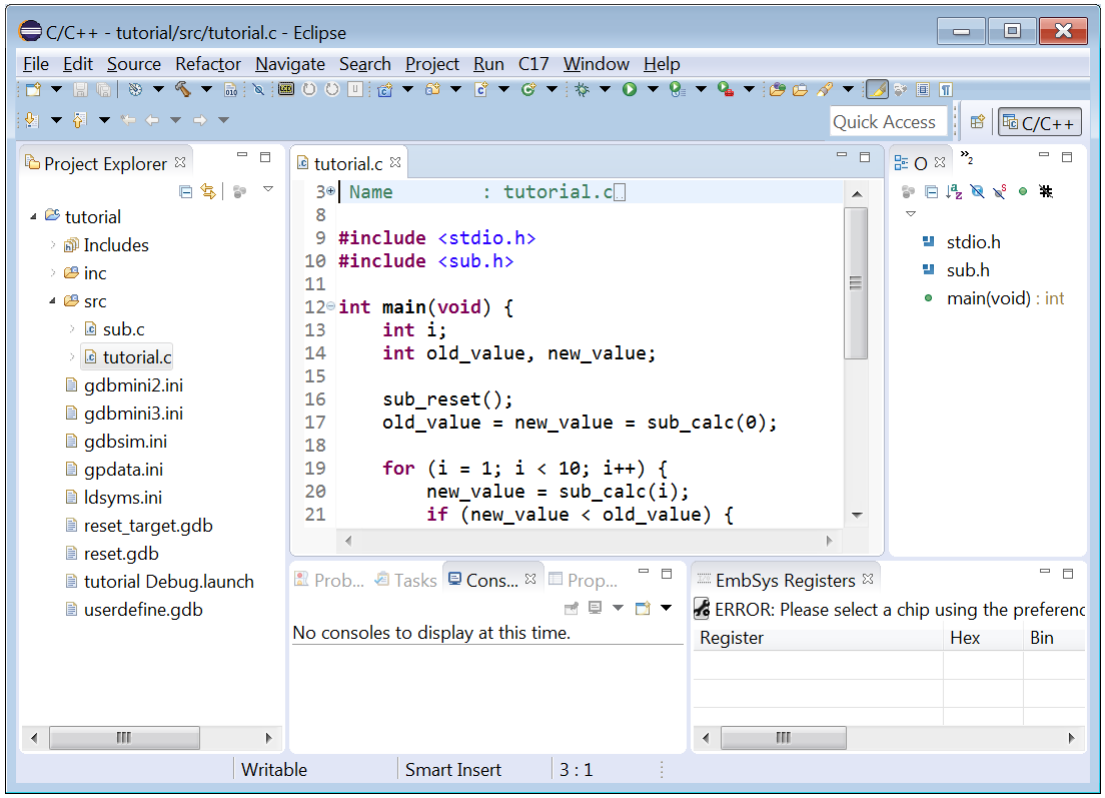


2.3.3 Displaying and Editing Source Files

The source files that have been newly created or imported into a project can be displayed and edited using the IDE editor.

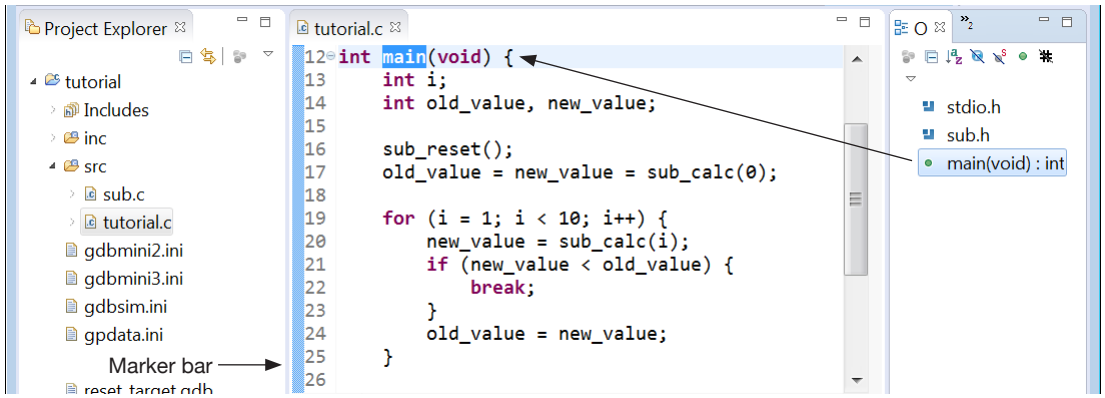
To edit a source file

Step 11: Double-click “tutorial.c” in the [Project Explorer] view.



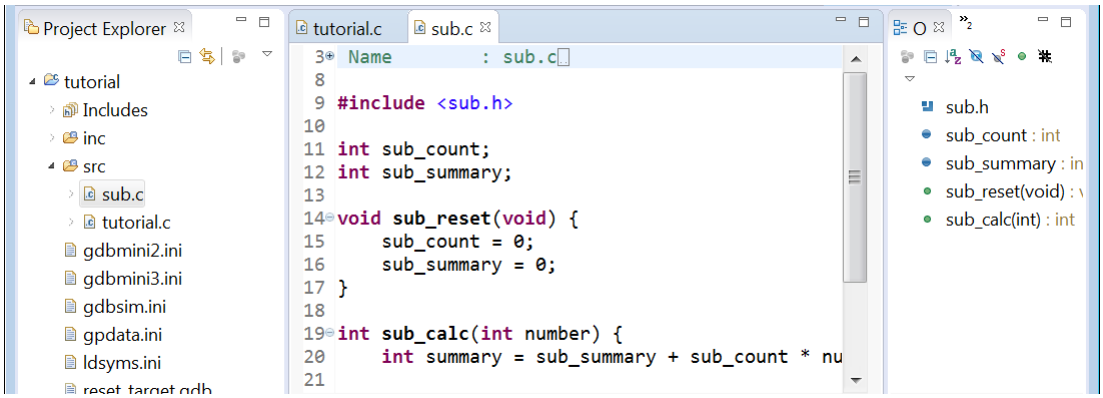
The contents of “tutorial.c” are displayed in the editor and they can be edited as with a general-purpose editor. For the editor functions and operation method, refer to the Eclipse help or a general book that describes Eclipse IDE for C/C++ Developers Package.
If a C source is displayed, reserved words of C language, comments, and strings are highlighted with colored characters. Furthermore, the IDE can be configured so that the selected file will be opened with a general-purpose editor usually used.

Step 12: Click “main(void): int” in the [Outline] view.



The editor will jump to the line where main() exists and highlight it. Furthermore, a bar indicating the range of the main() function will be displayed in the marker bar on the left side of the editor view. This allows check of functions and other elements easily.

Step 13: Double-click “sub.c” in the [Project Explorer] view.



Multiple sources can be opened at the same time. To select the source to display or edit, click the tab at the top of the editor view (where a file name is displayed).

Step 14: Click the ⊠ (Close) button on the editor tab of each source opened to close the editor.

Program startup processing

This package includes the startup processing library “crt0.o” in which a vector table and the boot and terminate functions that will be called before and after the main function are defined. This object will always be linked unless otherwise specified.

The startup processing library initializes the RAM and standard libraries, and enables interrupts, therefore, it is not necessary to implement these functions in the main function. If the startup processing contents must be changed, define custom processing functions in conforming to the specifications of the startup processing library.

For more information, refer to the IDE and library chapters in the “S1C17001C Manual.” Also refer to Section 3.2 in this manual, “Importing a GNU17 Ver. 2.x Project (When “crt0.o” is Used),” which describes about editing of sources according to “crt0.o.”

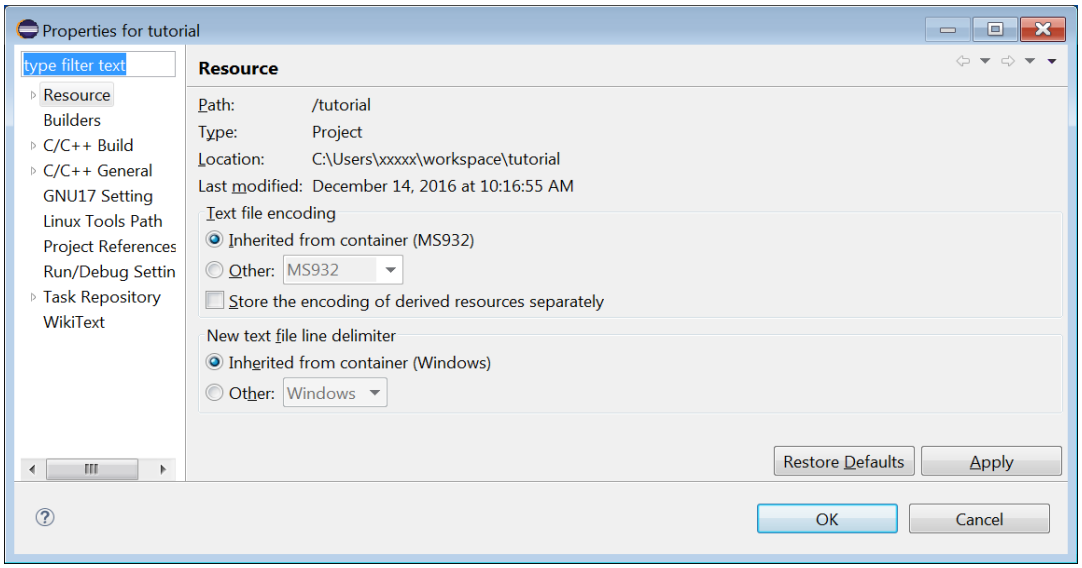
2.4 Basic Configuration of Project

The IDE needs various information (properties) to build a project correctly according to the target system. A page for setting basic properties is provided to configure most of all properties easily.
Normally, the basic properties can be set in the [GNU17 Project] wizard at new creation of a project as described in the "Creating a Project" section. They can also be changed later in the [Properties] dialog box of each project. This section shows how to set these properties using the [Properties] dialog box. For more information on each property, refer to Section 3.4.1, "Setting GNU17 Project Properties," in the "S1C17001C Manual."

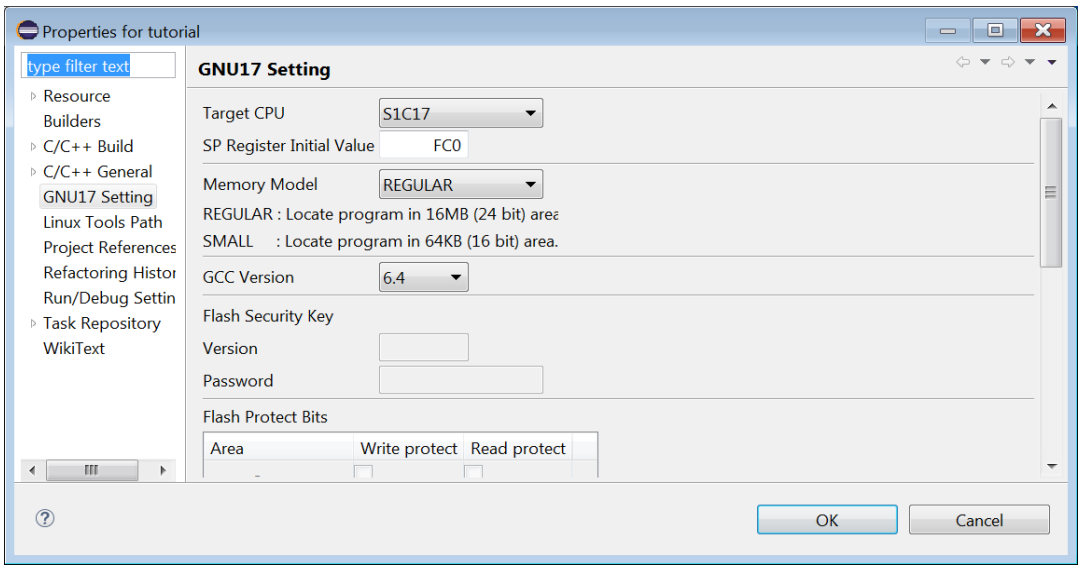
To open the basic configuration page

Step 1: Select "tutorial" in the [Project Explorer] view and select [Properties] from the [Project] menu (or context menu that appears by right-clicking).

This will bring up the [Properties] dialog box.



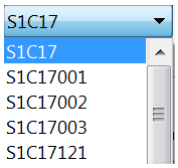
Step 2: Select [GNU17 Setting] from the property list.



Changing the basic properties

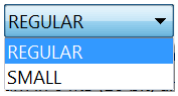
Step 3: Select the desired settings from the [Target CPU], [Memory Model], and [GCC Version] drop-down lists. It is not necessary to change settings in this tutorial.

Selecting target CPU



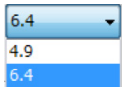
Select the model to be used.
Select [S1C17] when creating a model independent program.

Selecting memory model



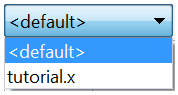
Select the address space supported by the model to be used.
← 24-bit address space (16MB)
← 16-bit address space (64KB)

Selecting GCC version



Select the version of the C compiler.
← gcc ver. 4.9
← gcc ver. 6.4

Selecting linker script



This drop-down list appears only when one or more linker script files exist in the project folder and is used to select the linker script file to be used. If a linker script file is selected here, it will be specified with the -T option when invoking the linker.
If <default> is selected, the default linker script for standard arrangement that has been incorporated in the linker will be used.
The "tutorial" folder does not contain a linker script file, so this drop-down list is not displayed.

* Linker script is information for the linker to arrange program codes and data to the specified memory locations. If the default linker script cannot build the user application, it is necessary to create a custom linker script file. For the contents of the linker script and how to create a linker script file using a wizard, refer to Appendix A, "Sections and Linker Script."

In addition to the properties shown above, the [GNU17 Setting] page allows settings of SP register initial value, flash security key, and flash protect bits (refer to the "Creating a Project" section).

Step 4: Click the [OK] button to terminate property settings (to close the dialog box) or click the [Apply] button to continue settings.

2.5 Project Configuration Details

Most properties are automatically configured from the project's basic configuration described in the previous section. These and all other properties can be configured in the [Properties] dialog box. Let's take a look at the settings pages here.
It is not necessary to change settings in this tutorial.

2.5.1 Environment Variable Settings

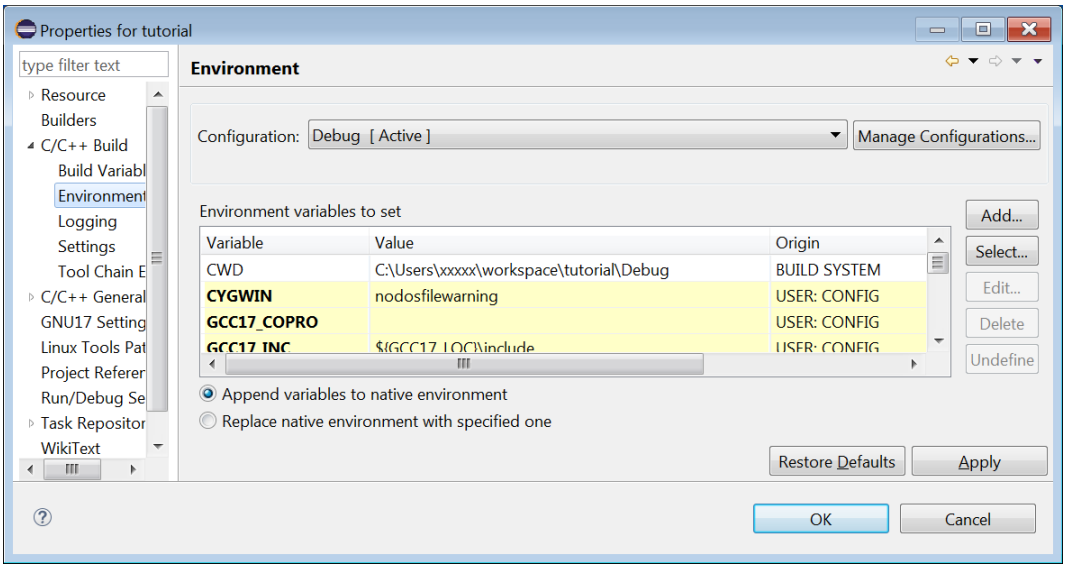
The IDE provides the environment variables to set the information required for building projects. The following shows a procedure to set them. For more information on the environment variables, refer to Section 3.4.2, "Setting Environment Variables," in the "S5U1C17001C Manual."

To open the environment variable setting page

Step 1: Select "tutorial" in the [Project Explorer] view and select [Properties] from the [Project] menu (or context menu that appears by right-clicking).

This will bring up the [Properties] dialog box.

Step 2: Select [C/C++ Build] > [Environment] from the property list.

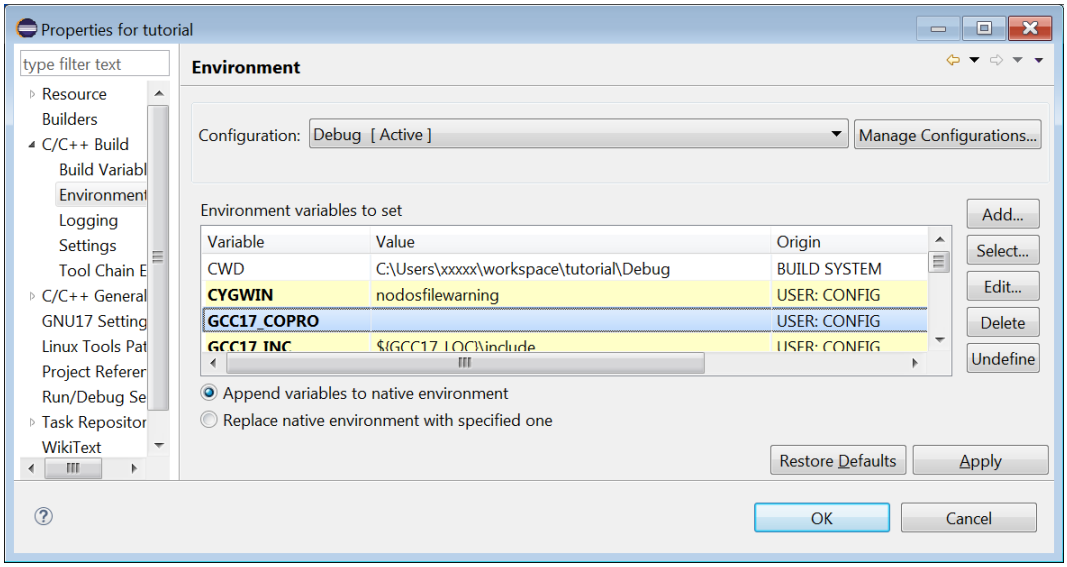


Most environment variables have been set according to the basic configuration of the project. The variables that may need to be set in this page are two items, "GCC17_COPRO" (coprocessor specification) and "GCC17_USER_LIBS" (user library specification).

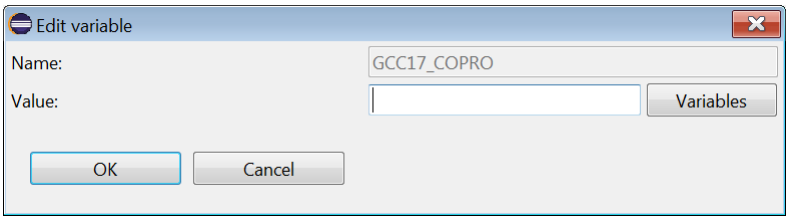
Specifying the coprocessor (GCC17_COPRO)

If a specific model has been selected for [Target CPU] of the basic configuration, it is not necessary to set "GCC17_COPRO," as it has been set correctly according to the model. To create a coprocessor specific program when "S1C17" is selected for [Target CPU], setting this environment variable as follows:

Step 3: Select the environment variable "GCC17_COPRO" and click the [Edit...] button.



This will bring up the [Edit variable] dialog box.



Step 4: Enter the symbol that represents the embedded coprocessor at [Value:] and click the [OK] button.

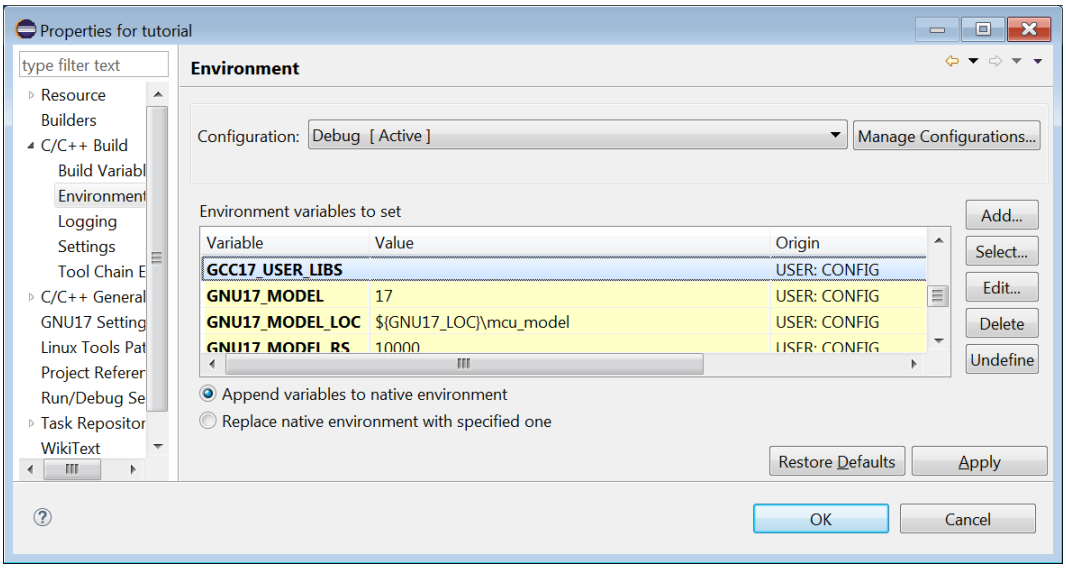
Table 2.5.1.1 GCC17_COPRO Settings

Coprocessor type	Setting
None	Blank (no input)
Multiplication coprocessor	\\M
COPRO	\\MD
COPRO2	\\MD2

Specifying a user library (GCC17_USER_LIBS)

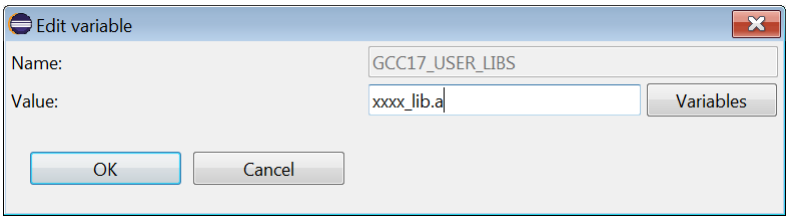
When linking a user library, the file name should be registered to this environment variable. In addition, the library file must be copied into the "(project)\Debug" folder.

Step 5: Select "GCC17_USER_LIBS" and click the [Edit...] button.



This will bring up the [Edit variable] dialog box.

Step 6: Enter the library name at [Value:] and click the [OK] button.



Two or more files can be specified by separating them with a semicolon.

2.5.2 Specifying Tool Options

Standard command options are specified for the build tools (C compiler, assembler, and linker) by default, but the command options can be added or deleted as necessary. Let's take a look at the setting page here.
It is not necessary to change the current setting except for the compiler optimization option in this tutorial.

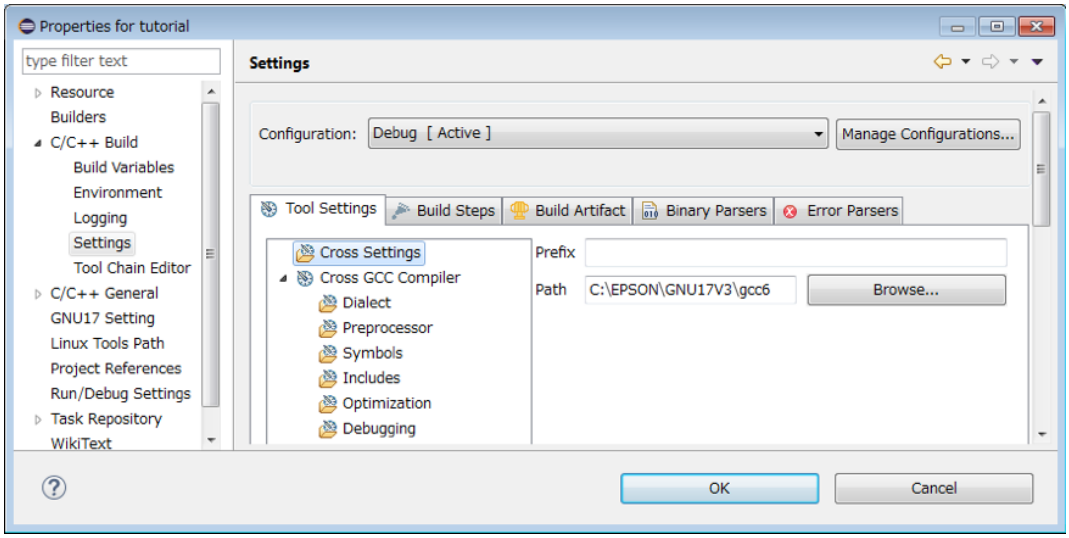
For detailed information on the command options for each tool, refer to Sections 3.4.3 to 3.4.6 in the "S5U1C17001C Manual."

To open the tool option setting page

Step 7: Select "tutorial" in the [Project Explorer] view and select [Properties] from the [Project] menu (or context menu that appears by right-clicking).

This will bring up the [Properties] dialog box.

Step 8: Select [C/C++ Build] > [Setting] from the property list to open the [Tool Settings] tab page.



Setting the path to the compiler

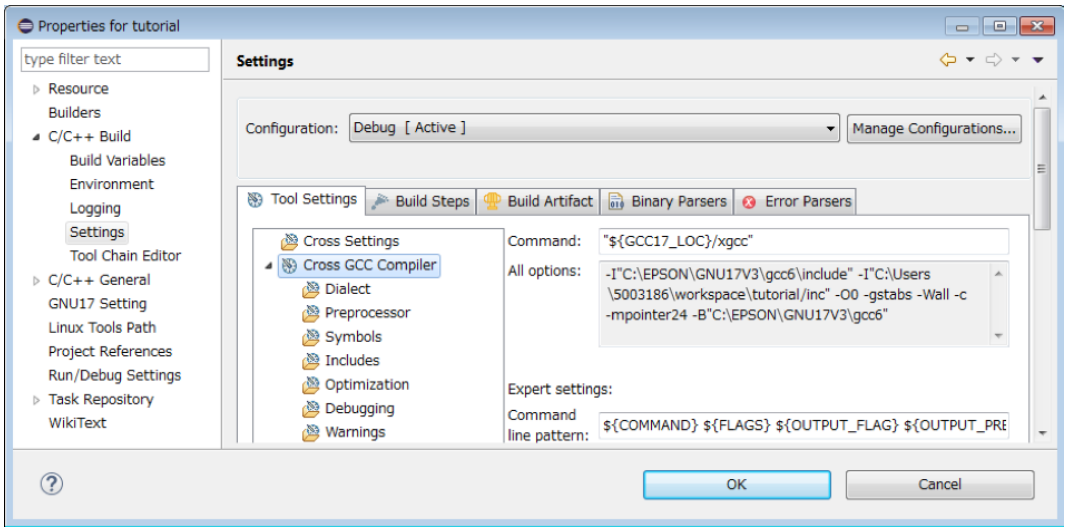
This page allows specification of the directory in which the C compiler and other tools to be used exist. Ordinarily, it is not necessary to change the setting here, as it has been set correctly according to the [GCC Version] selection in the basic configuration. For instance, when the tool directory has been moved to another location after being installed, the setting in this page should be corrected as follows:

Step 9: Select [Cross Setting] from the setting list in the [Tool Settings] page.

Step 10: Enter the full path of the tool directory at [Path] as shown in the screen above or select it using the [Browse...] button.

Setting compiler options

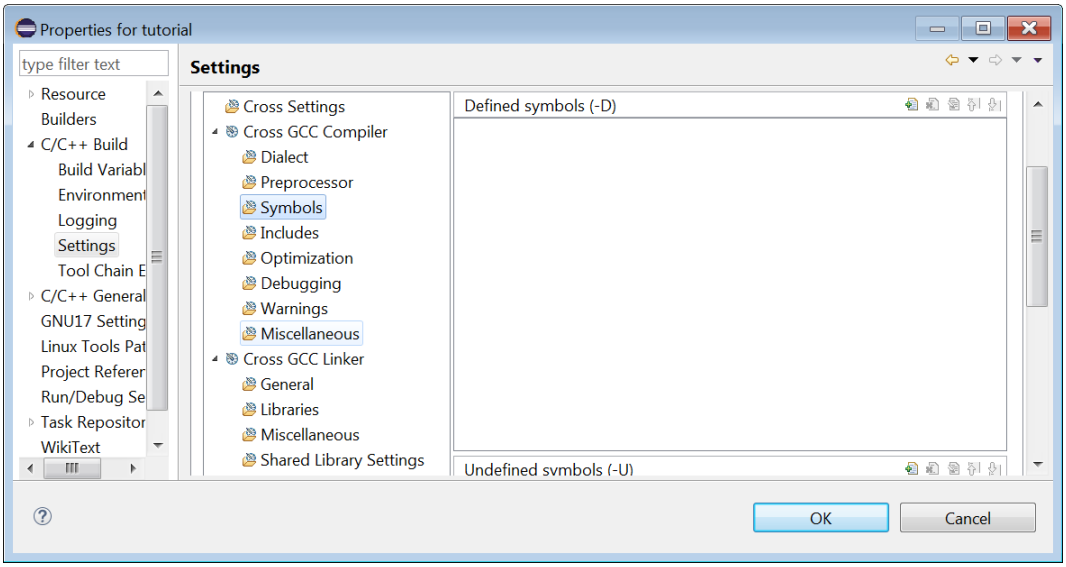
Step 11: Select [Cross GCC Compiler] from the setting list in the [Tool Settings] page.



The [All options:] shows the C compiler command options currently set.
If this setting should be changed, select an option from the setting list as shown below to open the option setting page.

Macro definition option (-D)

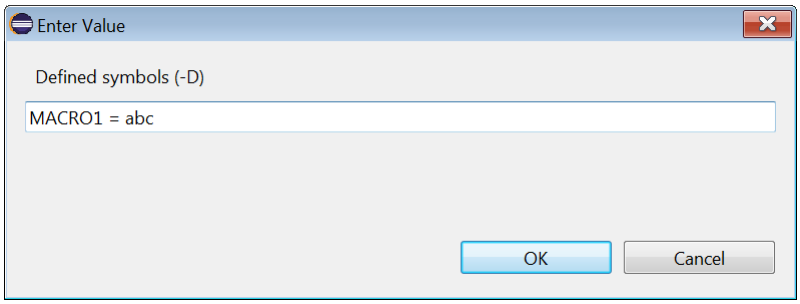
Step 12: Select [Symbols] from the list under [Cross GCC Compiler].

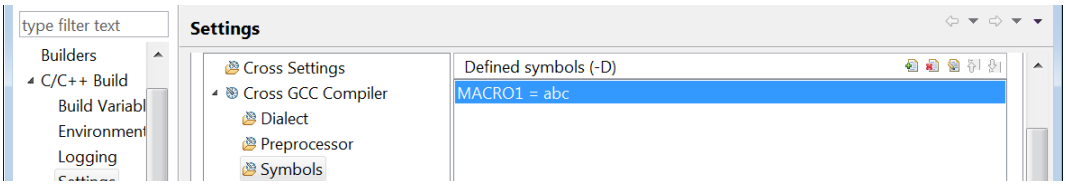


This page allows definition of macro names (specification of the -D option).

Step 13: Click the (Add...) button to bring up the [Enter Value] dialog box and enter the macro name to be defined, in a format shown below at [Defined symbol (-D)] ("-D" is not necessary to enter). Then, click the [OK] button.

1. <Macro name> (This will be defined as "<Macro name> = 1.")
2. <Macro name> = <Substitution characters>

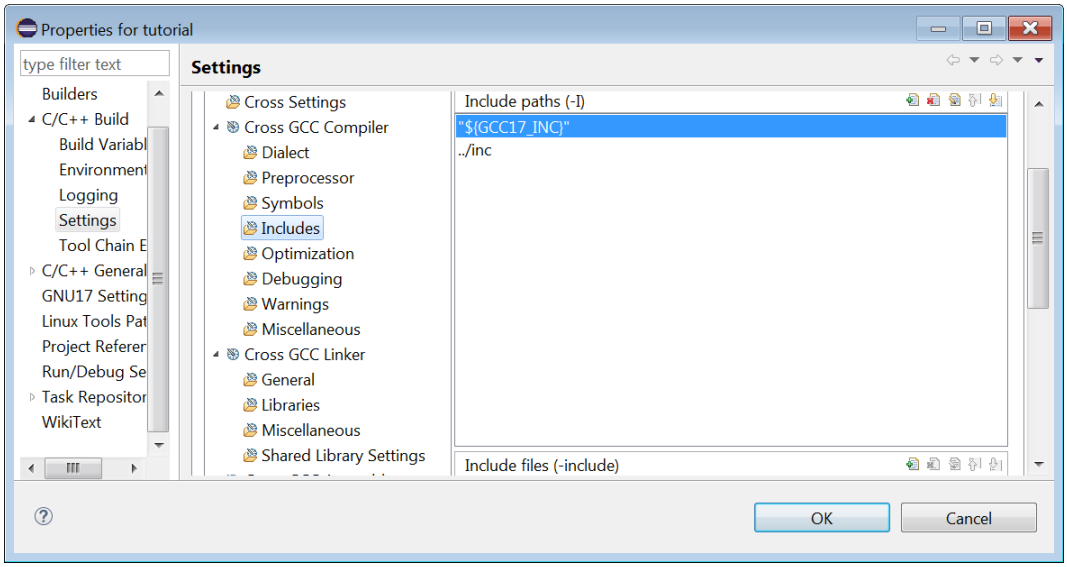




To define two or more macro names, repeat Step 13 for the number of macros.
No macro name has been defined by default.

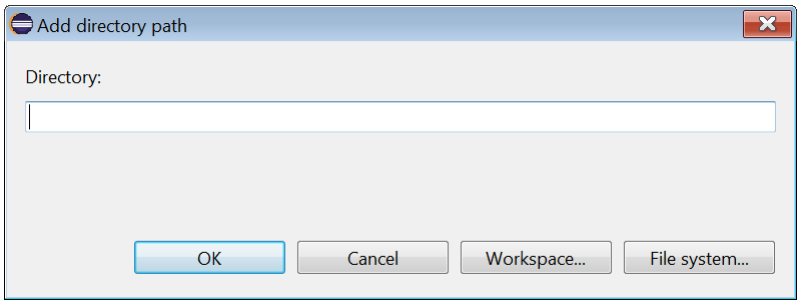
Include file directory specification option (-I)

Step 14: Select [Includes] from the list under [Cross GCC Compiler].



This page allows specification of include file directories (specification of the -I option).

Step 15: Click the (Add...) button to bring up the [Add directory path] dialog box.



Step 16: Click the [Workspace...] or [File system...] button to bring up the folder selection dialog box and select the include folder to be added from the workspace folder or another folder, respectively.

By default, the "${GCC17_INC}" (C:\GNU17V3\gcc6\include) and "(project)\inc" directories are specified.

Macros and environment variables for specifying paths

The "${GCC17_INC}" set by default is the macro that represents the path to the ANSI C library directory. GCC17_INC is the environment variable that has been defined as "${GCC17_LOC}/include" in the [Environment] page. Similarly, the path to the directory in which the GNU17 tools were installed has been defined as GCC17_LOC.

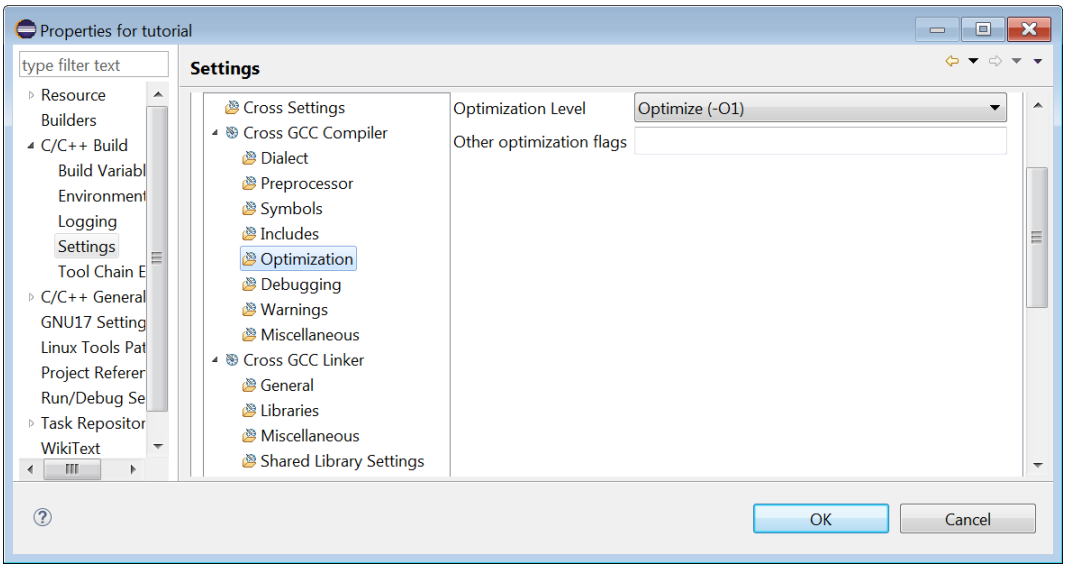
Example: When the GNU17 tools were installed in the "C:\EPSON\GNU17V3 directory

GCC17_LOC = C:\EPSON\GNU17V3\gcc6

GCC17_INC = C:\EPSON\GNU17V3\gcc6\include

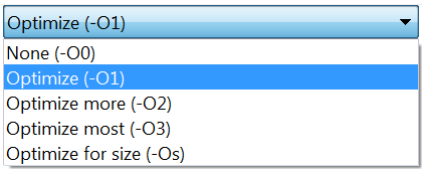
Optimization option (-O)

Step 17: Select [Optimization] from the list under [Cross GCC Compiler].



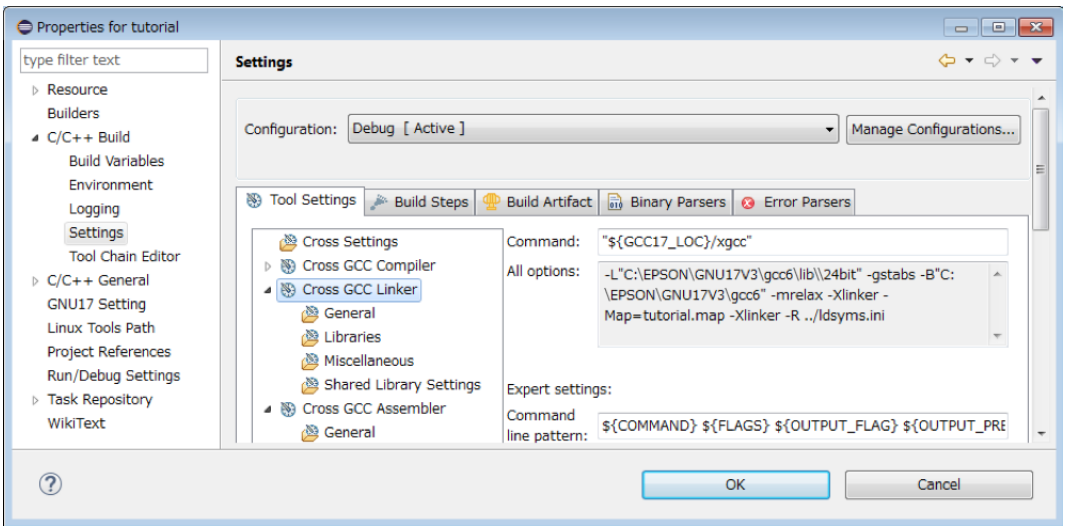
This page allows specification of an optimization level. The default setting is -O1.

Step 18: Select an optimization level from the [Optimization Level] drop-down list. Select -O0 (Not optimized) here in this tutorial for easy debugging.



Setting linker options

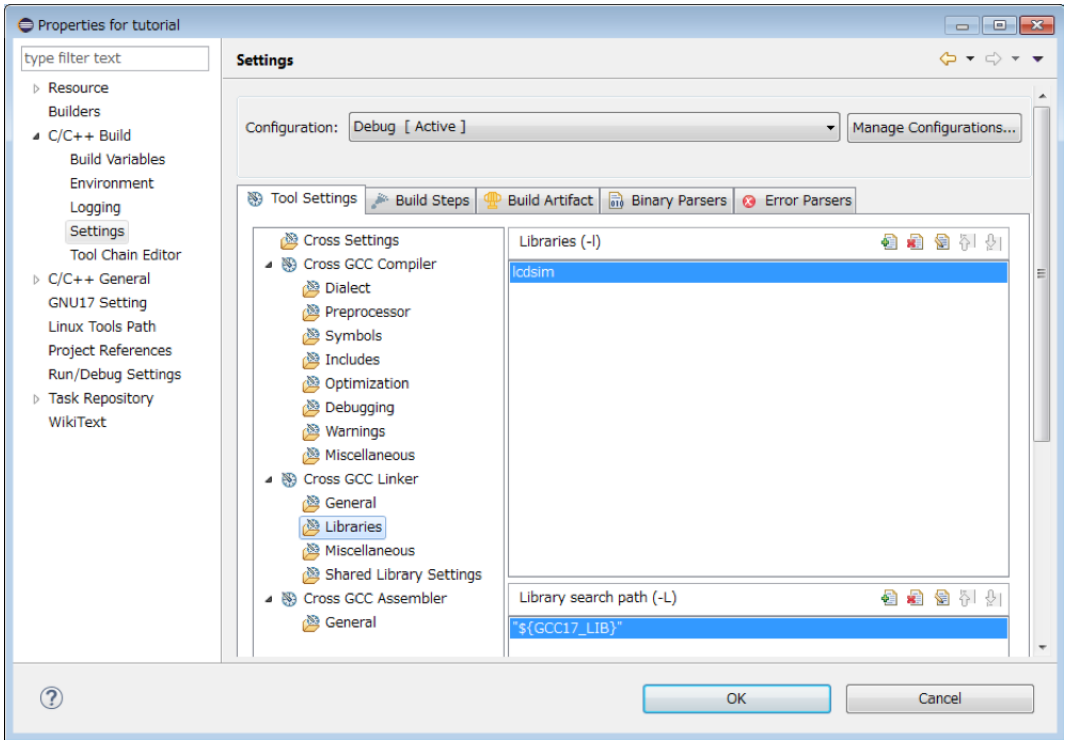
Step 19: Select [Cross GCC Linker] from the setting list in the [Tool Settings] page.



The [All options:] shows the linker command options currently set.
If this setting should be modified, select an option from the setting list as shown below to open the option setting page.

Specifying libraries

Step 20: Select [Libraries] from the list under [Cross GCC Linker].



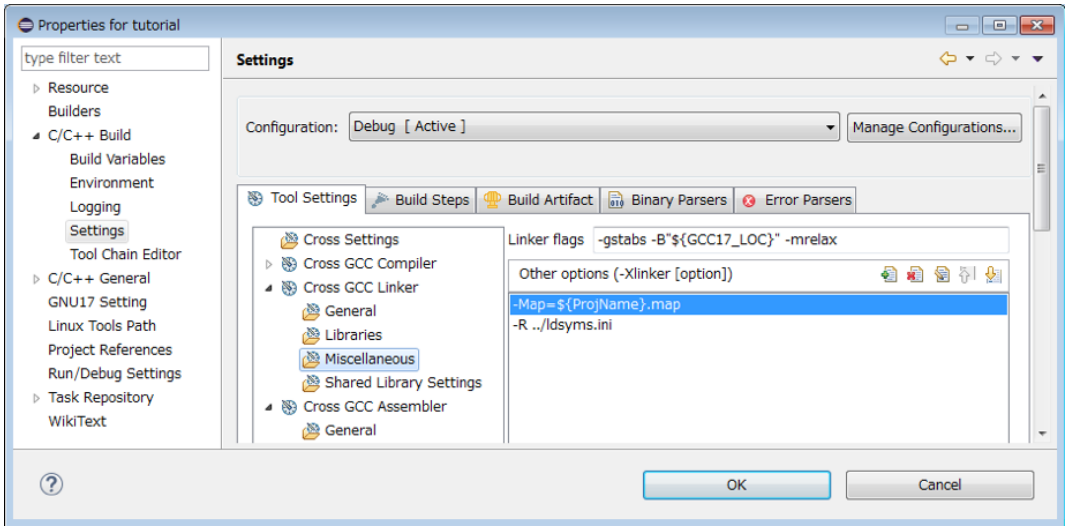
It is not necessary to specify the S1C17 library file here, as it will be linked by default. In this page, the LCD panel simulator library (lcdsim) has been specified so that it will be linked by default. The user library should be specified with an environment variable.

Step 21: Click the (Add...) button to bring up the [Enter Value] dialog box and enter a library name (xxx part of libxxx.a) at [Libraries (-l)]. Then click the [OK] button.

Library search paths should be set at [Library search path (-L)]. The ANSI C and emulation library directories have been defined as a macro called ${GCC17_LIB}, therefore, it is not necessary to change normally.

Specifying other linker command options

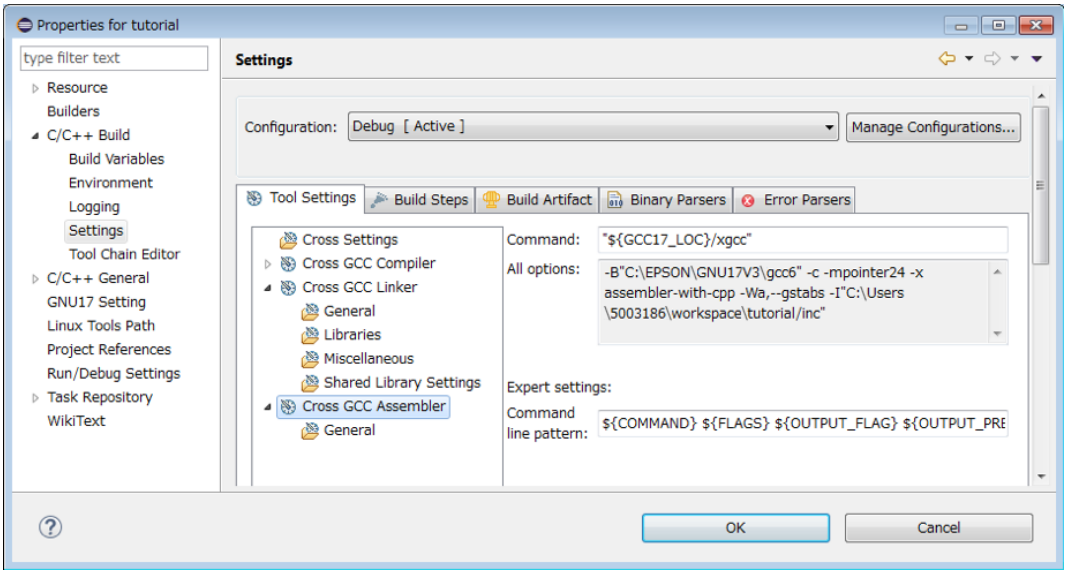
Step 22: Select [Miscellaneous] from the list under [Cross GCC Linker].



Linker command options can be specified in this page. The option to generate a map file ((project name).map) has been registered in advance.

Setting assembler options

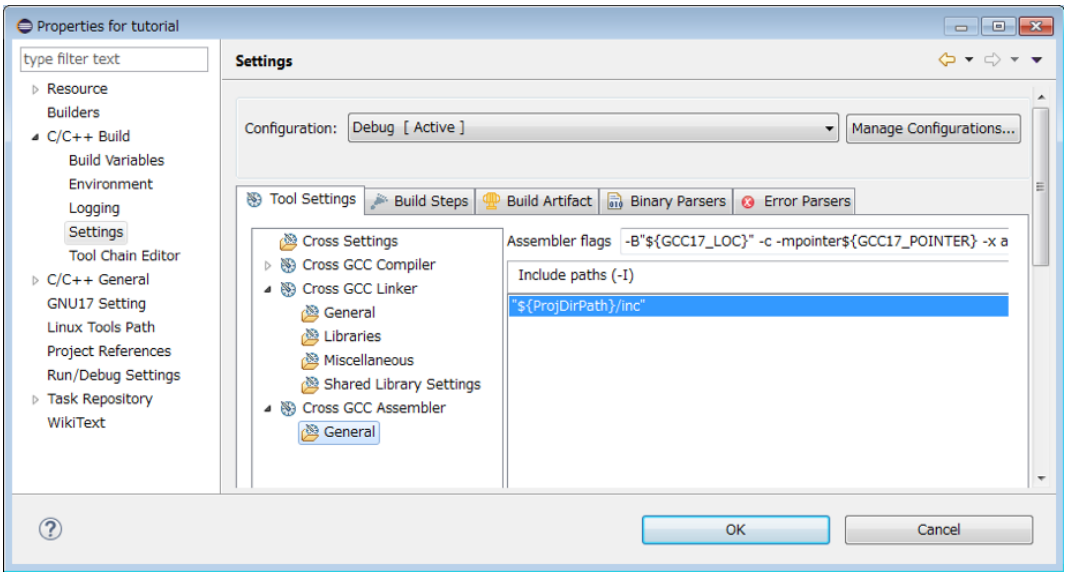
Step 23: Select [Cross GCC Assembler] from the setting list in the [Tool Settings] page.



The [All options:] shows the assembler command options* currently set.
If this setting should be modified, select an option from the setting list as shown below to open the option setting page.

* The IDE specifies the "-c -xassembler-with-cpp" option to assemble assembler sources through the C preprocessor.

Step 24: Select [General] under [Cross GCC Assembler].



Step 25: Enter the options required at [Assembler flags] and click the [OK] button.

The tool option settings are now complete.

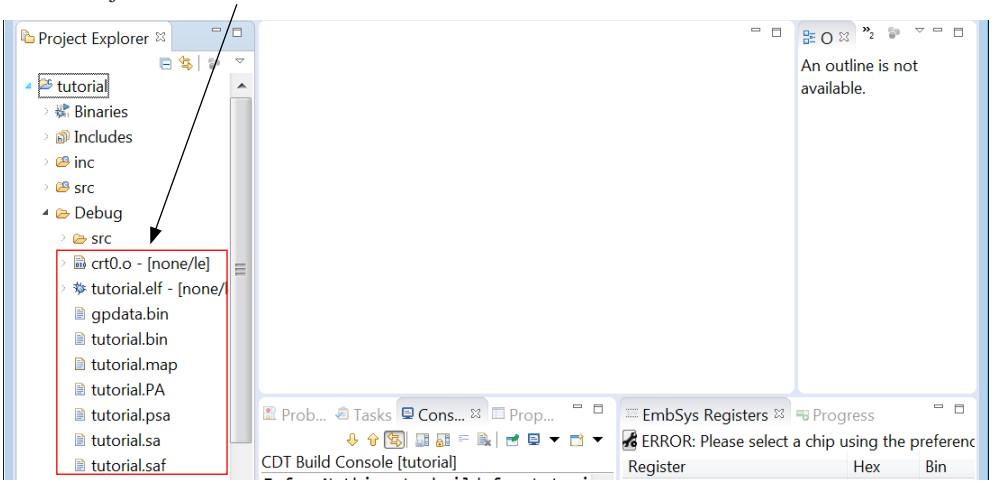
2.6 Building a Program

When the procedure described in the preceding sections has been finished, the program can be built (compiled, assembled, and linked).

To execute a build process

Step 1: Select “tutorial” in the [Project Explorer] view and select [Build Project] from the [Project] menu (or context menu that appears by right-clicking).

This will execute a build process and the tools that follow the linker to generate all the required files such as the executable object file “tutorial.elf” and submission file “tutorial.PA.”



∗1 Selecting [Build All] from the [Project] menu can build all the projects in the workspace.

∗2 When the build process is executed again without the source and header files modified after being executed once, the object and other output files are not regenerated. To regenerate the output files in this case, it is necessary to execute a clean process to erase the object file. For more information, refer to Section 3.5.3, “Clean and Rebuild,” in the “S5U1C17001C Manual.”

2.7 Debug

Debugging the program can be started by loading "tutorial.elf" after being generated in the build process. This section shows a basic debugging operation method.

2.7.1 Debugging Environment (ICDmini mode and Simulator mode)

The IDE supports two debugging environments (ICDmini mode and simulator mode).

ICDmini mode

In this mode, debugging can be performed by connecting the target system, which includes the actual MCU, to the debugger on the PC via the ICDmini (S5U1C17001H*). The target program is executed by the MCU on the target system, so hardware operation can be checked as well as software debugging.

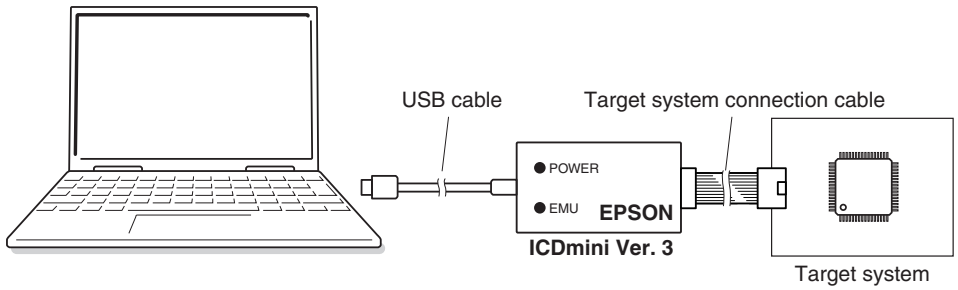


Figure 2.7.1.1 Example of Debugging System Using ICDmini3

Note: When debugging a program in ICDmini mode, the target CPU (basic configuration of project) must be selected correctly.

Simulator mode

This mode simulates target program execution in the PC memory without using other tools. However, the ICDmini-dependent function cannot be used.

This tutorial describes debugging operations in simulator mode.
For more information on the debugging environment, refer to the "S5U1C17001C Manual."

2.7.2 Preparation for Debugging (Selecting/Editing a GDB Command File)

Before debugging can be started, it is necessary to specify a GDB command file to be executed at startup of the debugger. For the debugger commands, refer to Section 8.5, "Command Reference," in the "S5U1C17001C Manual."

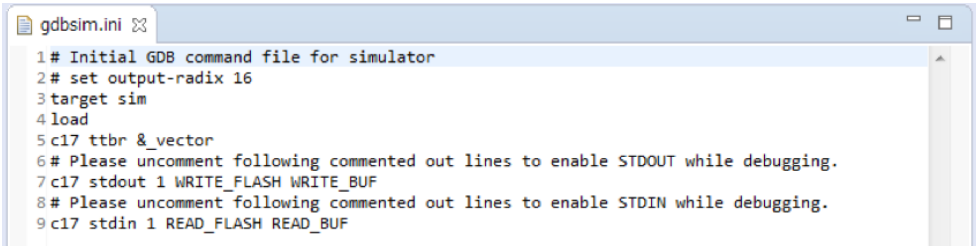
Auto-generated GDB command file

When a new project is created, the following three GDB command files for starting debugger are generated in the project folder.

gdbmini2.ini	GDB command file for ICDmini mode (for ICDmini Ver. 1–2)
gdbmini3.ini	GDB command file for ICDmini mode (for ICDmini Ver. 3)
gdbsim.ini	GDB command file for simulator mode

To see the contents of these files, display them in the editor.

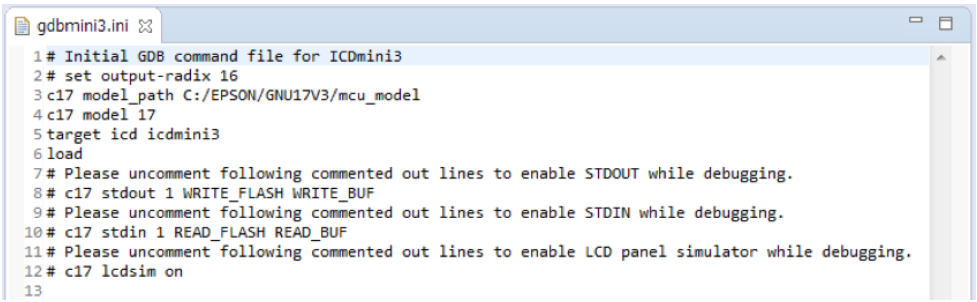
Step 1: Double-click on “gdbsim.ini” under the “tutorial” project in the [Project Explorer] view.



The contents of the GDB command file for simulator mode will be displayed in the editor. This GDB command file executes the operations shown below.

1. Sets the debugger into simulator mode.
2. Loads the program to the target (simulator) memory.
3. Sets the TTBR register of the CPU.
4. Enables the standard I/O function (stdout/stdin).

Step 2: Double-click on “gdbmini3.ini” under the “tutorial” project in the [Project Explorer] view.



The contents of the GDB command file for ICDmini mode will be displayed in the editor. This GDB command file executes the operations shown below.

1. Specifies the target (MCU).
2. Sets the debugger into ICDmini mode.
3. Loads the program to the target memory.

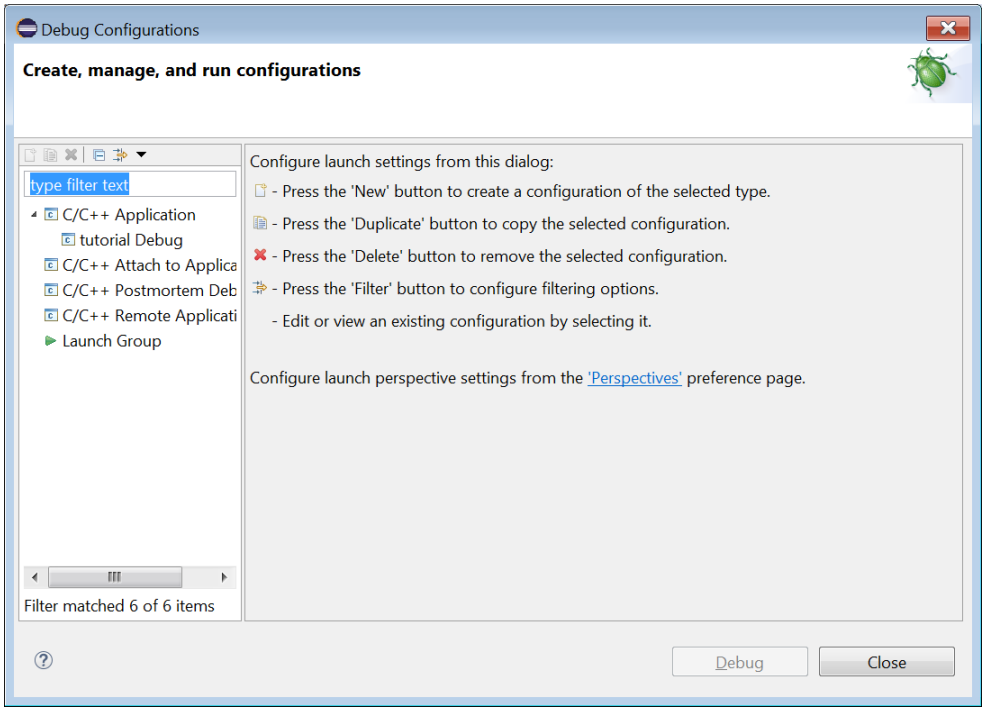
Selecting a GDB command file

This tutorial uses “gdbsim.ini” for debugging. To execute this GDB command file at startup of the debugger, configure the debugger as follows in advance.

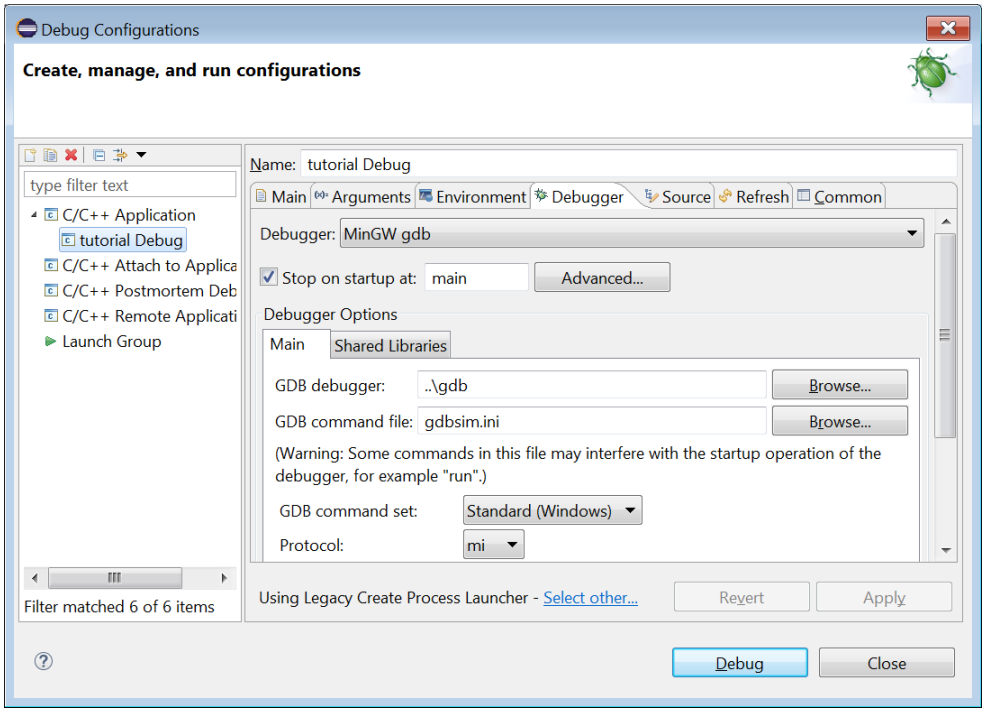
Step 3: Select “tutorial” in the [Project Explorer] view and select [Debug Configurations...] from the (Debug) drop-down list* in the toolbar.

* [Debug Configurations...] can also be selected from the [Run] menu.

This will bring up the [Debug Configurations] dialog box.



Step 4: Select [C/C++ Application] > [tutorial Debug] from the list and open the [Debugger] tab page.



Step 5: Select the GDB command file to be executed using the [Browse...] button to set it at [GDB command file:]. In this tutorial, leave “gdbsim.ini” unchanged.

A user created GDB command file can also be selected here.

Editing a GDB command file

In simulator mode, it is not necessary to edit "gdbsim.ini" particularly. In ICDmini mode, modify the underlined parts shown below as necessary.

```
# Initial GDB command file for ICDmini3
# set output-radix 16
c17 model_path C:/EPSON/GNU17V3/mcu_model        ... (1)
c17 model 17xxx                                  ... (2)
target icd icdmini3                              ... (3)
load
# Please uncomment following commented out lines to enable STDOUT while debugging.
# c17 stdout 1 WRITE_FLASH WRITE_BUF             ... (4)
# Please uncomment following commented out lines to enable STDIN while debugging.
# c17 stdin 1 READ_FLASH READ_BUF
# Please uncomment following commented out lines to enable LCD panel simulator while debugging.
# c17 lcdsim on                                  ... (5)
```

(1) c17 model_path

Specify the path to the directory in which the device information definition files are stored. It is not necessary to modify if the location has not been changed from the installed directory.

(2) c17 model

Specify the target model name. The IDE writes the model name, that was specified when the project was newly created or specified as the basic configuration of the project, here when it creates the GDB command file. It is not necessary to modify other than when required for debugging.

If nothing is connected to the TARGET VCC IN pin of the ICD, the model name should be followed by @ NOVCCIN as a Detail option. For the Detail options that can be specified, refer to the document attached to the device information definition file (fls/fls17*_readme.txt).

In simulator mode, specifying a target model name will invoke the peripheral circuit simulator (ES-Sim17) when starting the debugger if the model supports it.

(3) target

Specify the debug mode. When ICDmini mode is specified, a target system and an ICDmini must be connected to the PC. Normally, it is not necessary to modify this parameter, as is set properly by using an auto-generated GDB command file that meats the debugging environment.

target sim	Simulator mode This parameter is described in "gdbsim.ini."
target icd icdmini3	ICDmini mode (when ICDmini Ver. 3 is used) This parameter is described in "gdbmini3.ini."
target icd icdmini2	ICDmini mode (when ICDmini Ver. 1.0/1.1/2.0 is used) This parameter is described in "gdbmini2.ini."

(4) c17 stdout and c17 stdin

By uncommenting these lines (removing #), the contents output to stdout can be displayed in the [Console] view. And data can be input to stdin of the program through the dedicated input window.

(5) c17 lcdsim

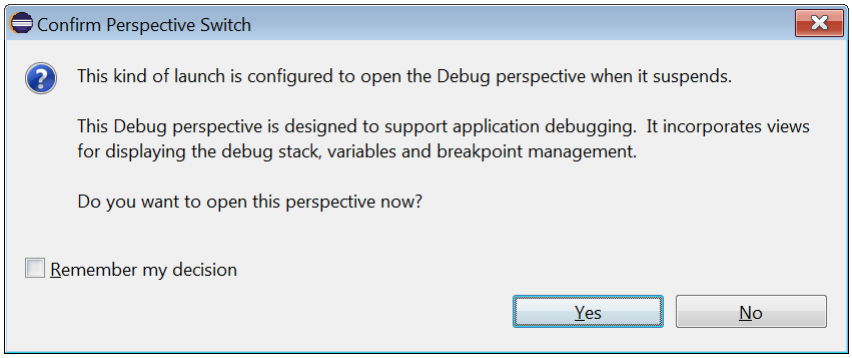
Uncommenting this line (removing #) enables use of the LCD panel simulator function.

When executing other commands at startup of the debugger, add them in the GDB command file selected.

2.7.3 Launching the Debugger

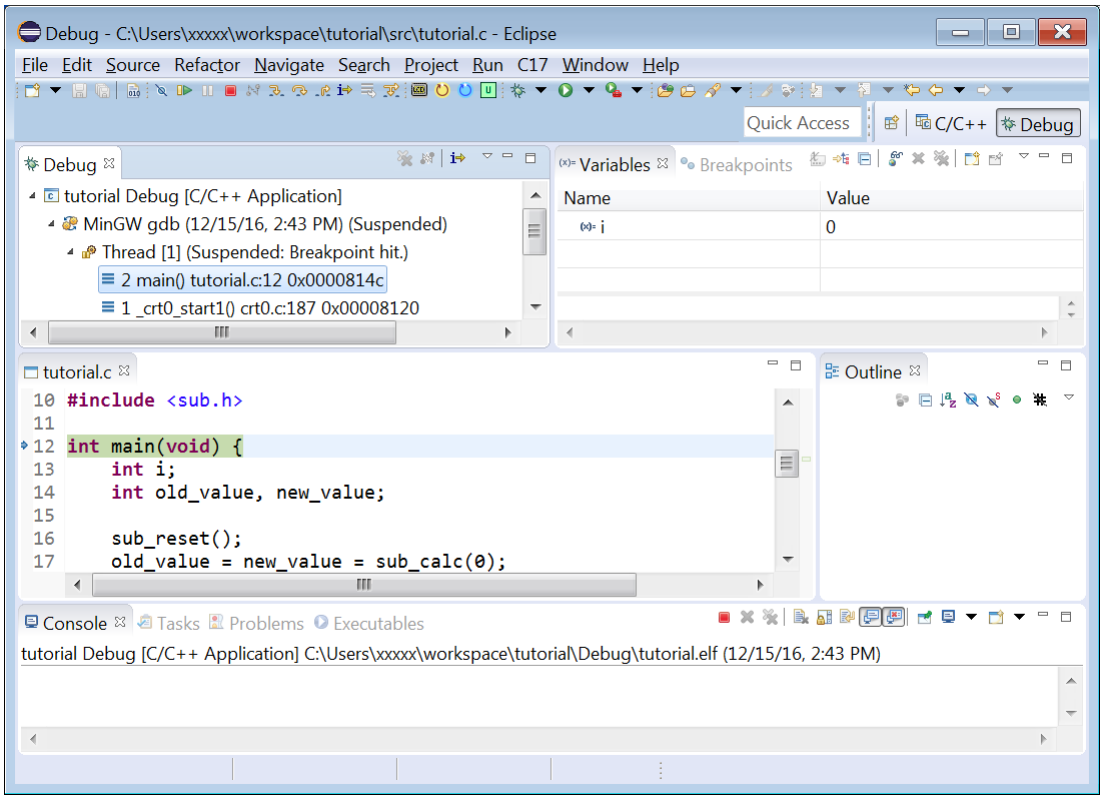
To launch the debugger

Step 6: Select [C/C++ Application] > [tutorial Debug] from the list in the [Debug Configurations] dialog box, and then click the [Debug] button. If the [Confirm Perspective Switch] dialog box appears, click the [Yes] button.



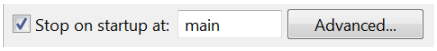
* This dialog box appears for confirming that the IDE perspective (view configuration) will be switched from [C/C++] to [Debug]. Selecting the [Remember my decision] checkbox disables this dialog box from appearing the next time.

The debugger starts up and the window is switched to the [Debug] perspective.



The object file “tutorial.elf” is loaded and the program execution is suspended at the main function (top of the function) after the boot process has been executed.
“2 main() tutorial.c: 12 0x0000814c” in the [Debug] view shows the breakpoint where the program execution has suspended.
The editor shows the suspended position by highlighting line 12 of “tutorial.c” in green.

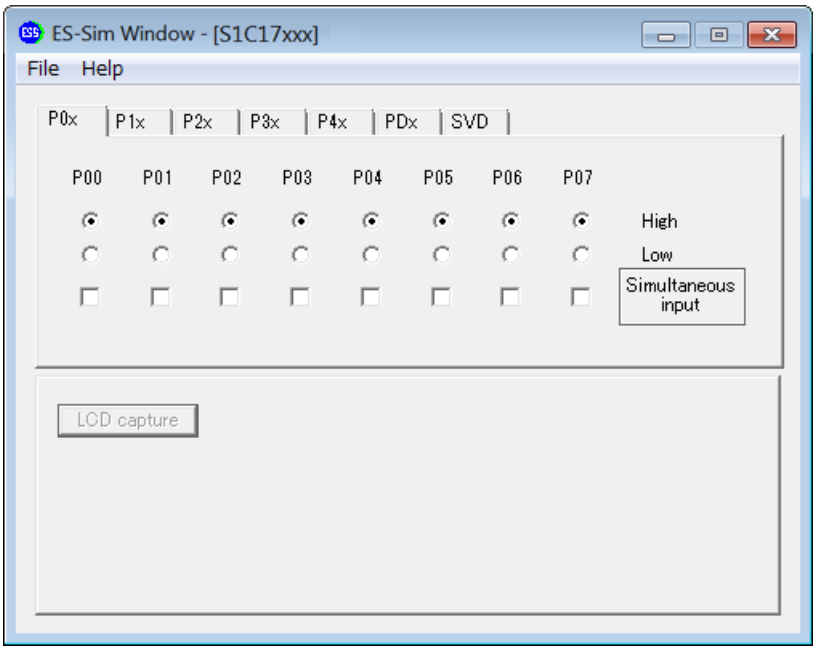
* The program execution has been suspended because the [Debug] tab page in the [Debug Configurations] dialog box has been set as follows:



Deselecting this checkbox enables the program to continue running without suspending after being started. Also a position other than the main function can be specified so that the program execution will be suspended at that position.

Peripheral circuit simulator

When the debugger is launched in simulator mode with a target CPU, which supports the peripheral circuit simulator (ES-Sim17), selected, the peripheral circuit simulator starts running as well.



In this window, GPIO port input/output, SVD operation, and LCD driver display statuses can be simulated. For more information, refer to “Peripheral Circuit Simulator (ES-Sim17)” in the “S5U1C17001C Manual.”
The models that can invoke the peripheral circuit simulator have the CPU configuration file “essim17” in the model information folder (GNU17V3/mcu_model/17xxx).

2.7.4 Debugger Toolbar Buttons Overview

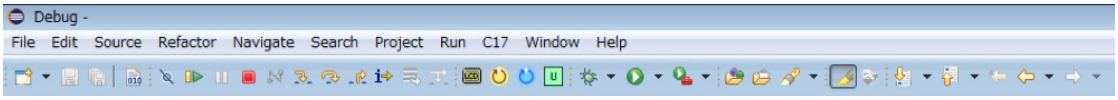


Table 2.7.4.1 Debugger Toolbar Buttons

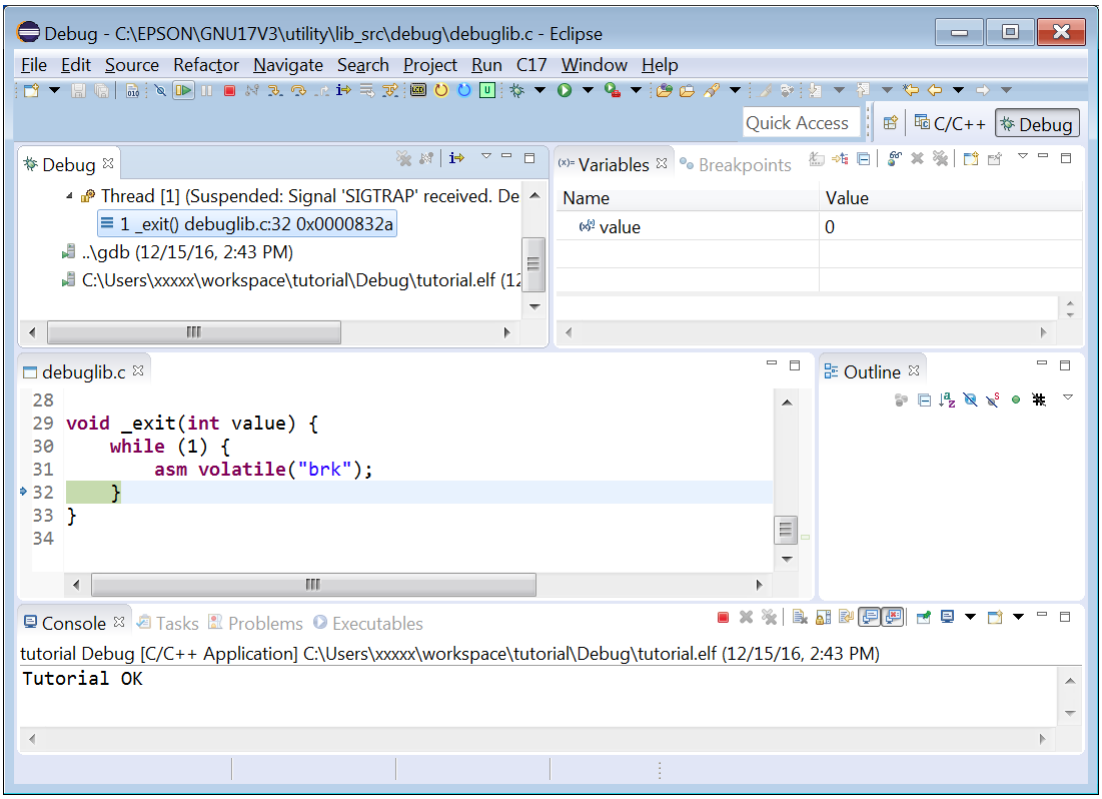
Icon	Name	Function
	[Skip All Breakpoints]	Ignores the breakpoints that have been set.
	[Resume]	Executes the program.
	[Suspend]	Suspends the program being executed. In debug mode, MCU internal information can be monitored and edited, and various items can be configured.
	[Terminate]	Terminates the GDB debugger.
	[Step Into]	Executes the current program step.
	[Step Over]	Executes the current program step including the sub-functions.
	[Step Return]	Executes the program steps until exiting the current function.
	[Instruction Stepping Mode]	The [Step *] commands will be executed in units of assembler instruction when this button is set to active.
	[Launch LCD Utility]	Opens the LCDUtil17 window.
	[Reset]	Jumps to the head of the program to initialize the general-purpose registers.
	[Reset Target]	Resets the target hardware.
	[User Command]	Executes a user defined command. Use the “userdefine.gdb” file to define commands.
	[Debug]	Launches the debugger.

2.7.5 Program Execution

To execute the program

Step 7: Click the (Resume) button (or select [Resume] from the [Run] menu).

The sample program starts running from the current position. It calculates the square pyramidal numbers (the number of stacked spheres that constitutes a quadrangular pyramid) from one to nine stories and displays "Tutorial OK!" in the [Console] view before being terminated.



The program stops by the brk instruction in the _exit function called after the main function has been completely executed, therefore, the editor opens the source and highlights the stop position.

To suspend the program being executed

The operations shown below can suspend the program being executed during actual software development.

Step 8: To suspend the program so that it will be able to resume, click the (Suspend) button in the tool bar (or select [Suspend] from the [Run] menu).

To resume the program from the suspended position, click the (Resume) button.

Step 9: To terminate the current debug session, click the (Terminate) button (or select [Terminate] from the [Run] menu).

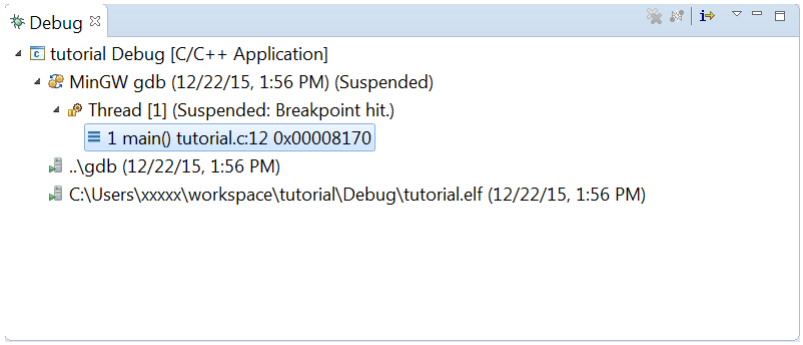
To restart debugging after that, click the (Debug) button (or select [Debug] from the [Run] menu).

2.7.6 Debugger Views

Let's take a look at the main views used for debugging here.

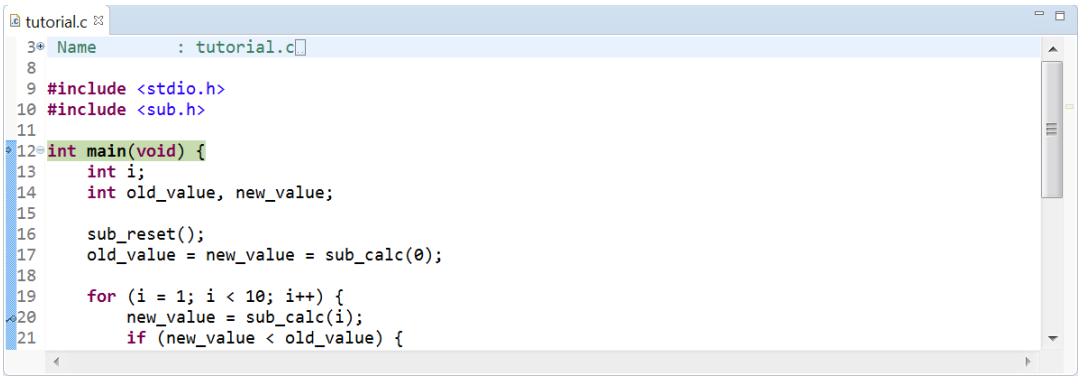
Step 10: Views that are not currently displayed can be opened by selecting them from the [Window] menu > [Show View] sub-menu.

[Debug] view



This view displays debug information.
In the example above, the displayed contents show that "tutorial.elf" has been loaded to the debugger gdb and the program execution has been suspended at the main function position in "tutorial.c" (address 0x8170) by hitting a breakpoint.

Editor view



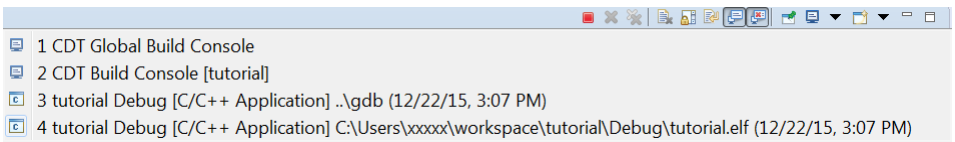
This view displays the source being executed or suspended. The current position where the program execution has been suspended is highlighted in green.
The arrow in the marker bar at the left side of the view indicates the current line pointed by the program counter and indicates that the line has been set to be a breakpoint.

[Console] view

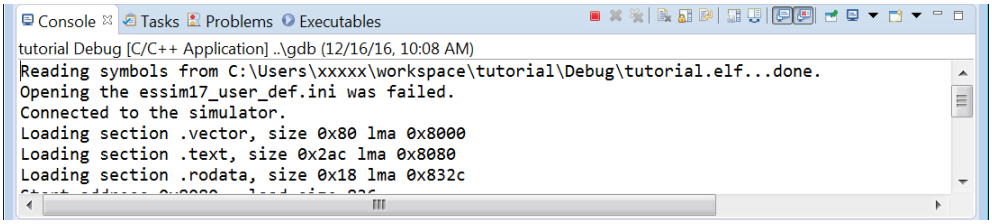
This view displays debugger messages and is also used as a standard input/output for the target program.

To display debugger messages

Step 11: Select "tutorial Debug [C/C++ Application] ..\gdb" from the (Display Selected Console) drop-down list.



Example of debugger message display

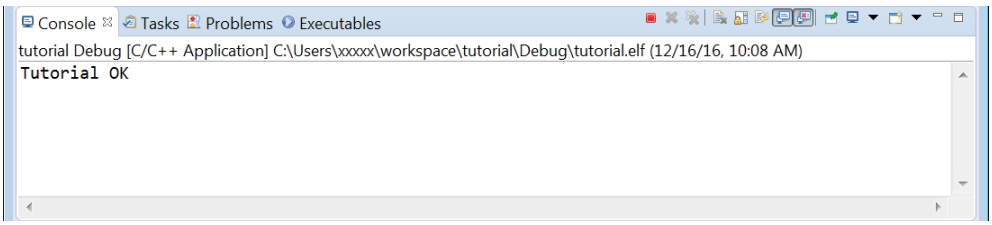


∗ The debugger console allows entering of debug commands to execute. No (gdb) prompt is displayed. Enter the command to be executed at the last line and then press the [Enter] key.

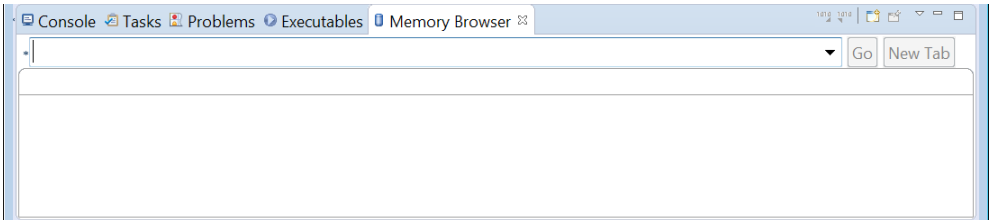
To use the [Console] view for input/output to/from the target program

Step 12: Select “tutorial Debug [C/C++ Application] C:\ ... \tutorial\Debug\tutorial.elf” from the (Display Selected Console) drop-down list.

Example of target program output



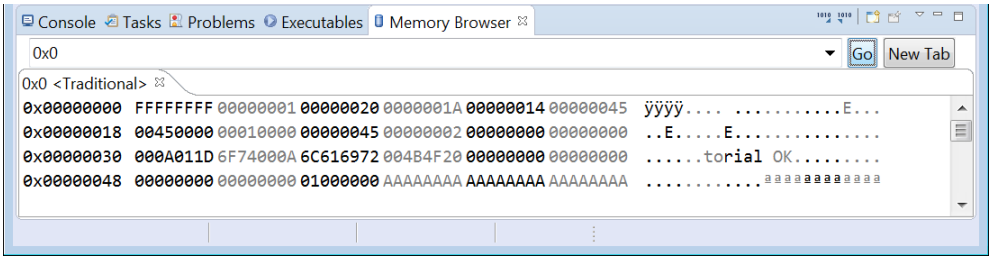
[Memory Browser] view



This view displays the contents of the target memory specified.

To specify the memory area to be displayed

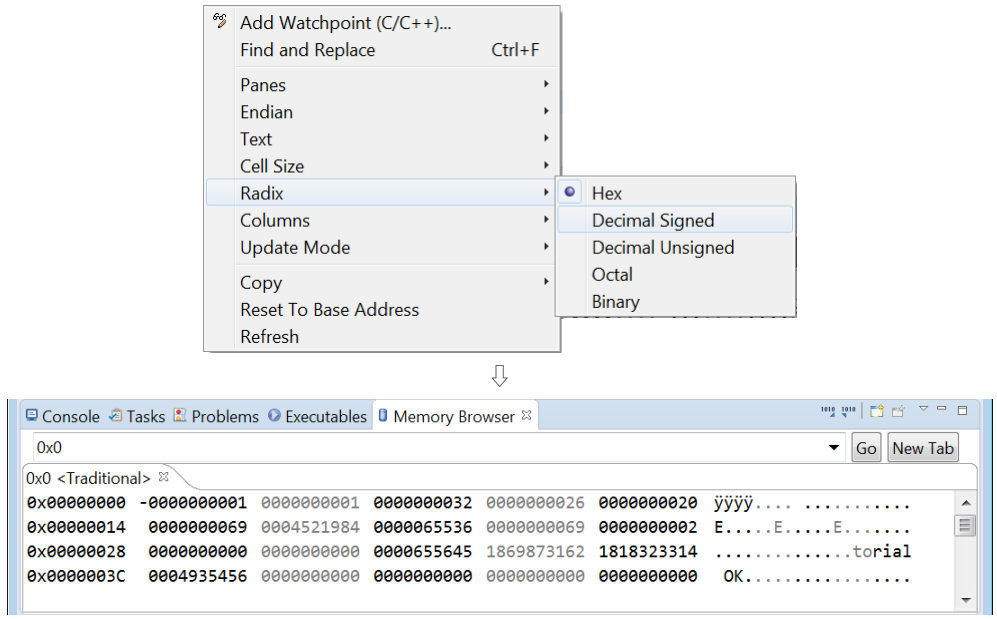
Step 13: Enter the memory address or variable name to be monitored into the text box and then click the [Go] button (or press [Enter]).



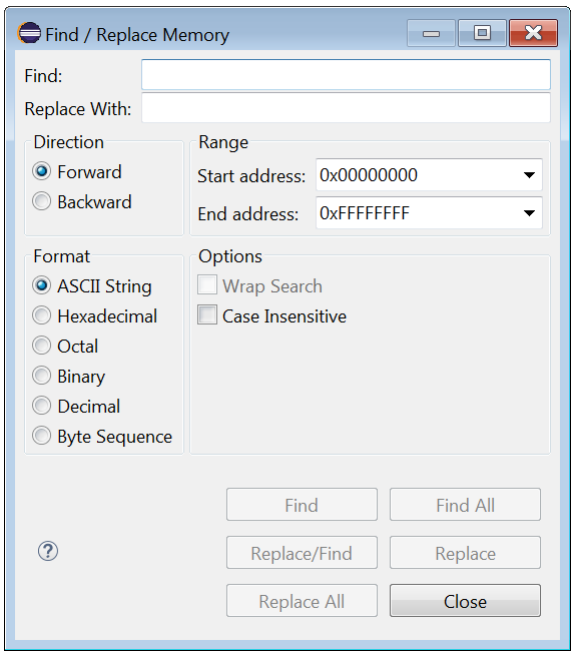
The memory contents are displayed from the specified address in hexadecimal dump format. The memory contents can also be edited by clicking the memory value and entering a numerical value from the keyboard. Clicking the [New Tab] button opens a new tab page. Use this when monitoring two or more areas.

To change the display format

Step 14: Right-click in the view to display a context menu and select the desired display format.



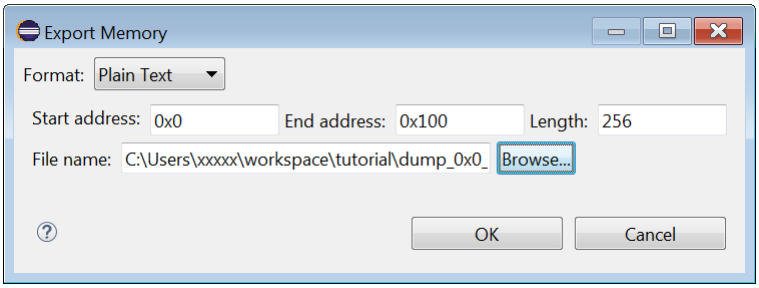
This example changes a HEX format into a signed decimal format. In addition to this, text encode format, number of columns, number of bytes in each column, and other conditions can be selected.
Also this context menu provides the [Find and Replace] menu item allowing retrieval/replacement of the specified number/character string.



To export memory data to a file

The specified memory area can be output to a file.

Step 15: Click the (Export) button to bring up the [Export Memory] dialog box.



Step 16: Set the output format (text, binary, or S record), address range, and file name, and click the [OK] button.

Output examples:

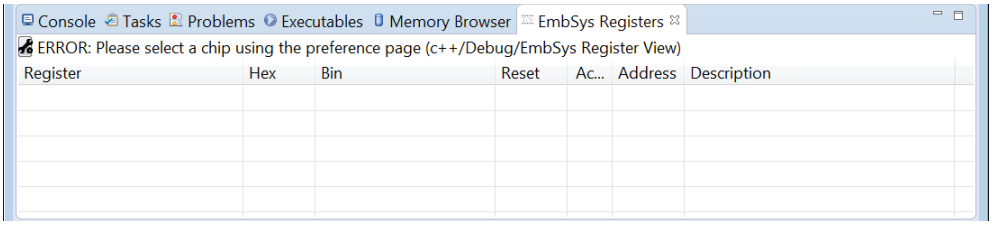
Text

```
FFFFFFFF 01000000 20000000 1A000000 14000000
45000000 00004500 00000100 45000000 02000000
00000000 00000000 0A001D01 0A00746F 7269616C
204F4B00 00000000 00000000 00000000 00000000
00000001 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA
AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA
    :        :        :        :        :
```

S record

```
S315000000000FFFFFFFF01000000200000001A000000B3
S3150000001014000000450000000000450000000001003B
S3150000002045000000020000000000000000000000083
S315000000300A001D010A00746F7269616C204F4B0043
S3150000004000000000000000000000000000000000AA
S3150000005000000001AAAAAAAAAAAAAAAAAAAAAAAA1
    :        :        :        :        :
```

[EmbSys Registers] view



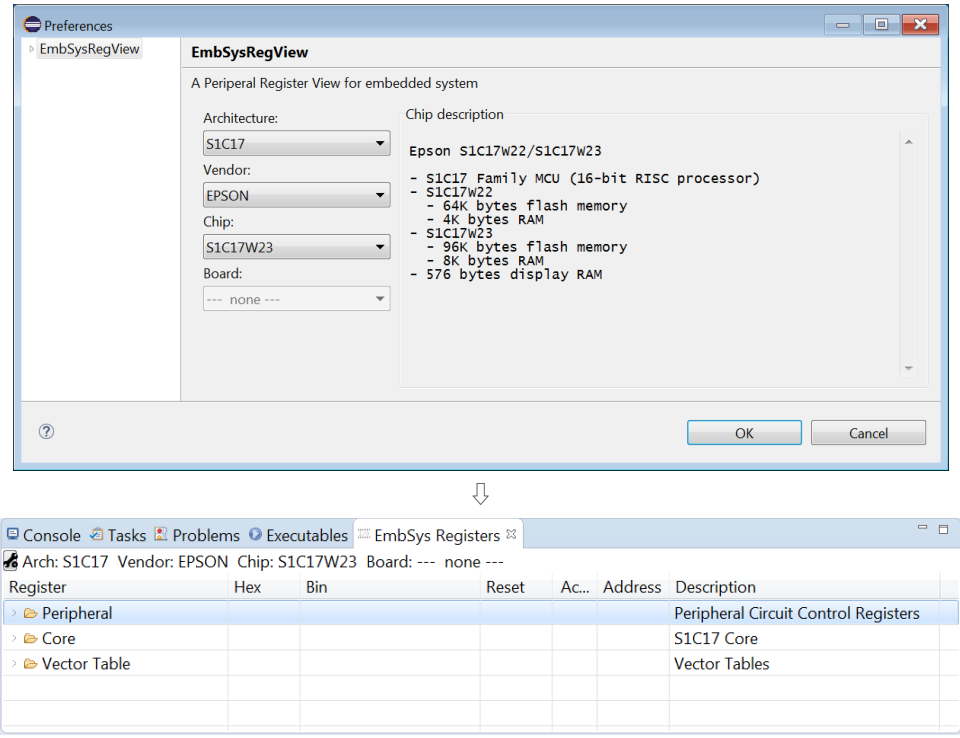
This view displays the contents of the memory-mapped core/peripheral circuit control registers and vector table.

To specify the model

A model must be selected for this view to display the contents.

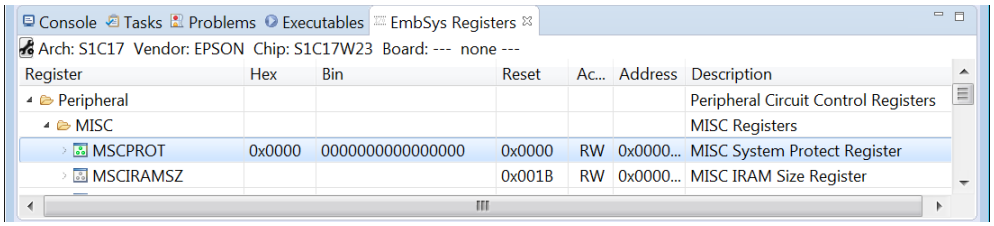
Step 17: Click the button to bring up the [Preferences] dialog box and set the selection items as below. Then click the [OK] button to close the dialog box.

Architecture: S1C17
Vendor: EPSON
Chip: Model name (e.g. S1C17W23)



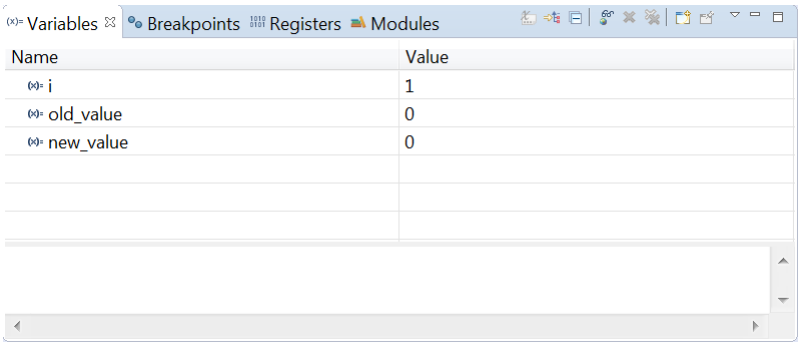
To display current register values

Step 18: Expand [Peripheral] > [(Peripheral circuit)] to display the register to be monitored. Clicking the icon on the left of the register name enables the register value to be displayed in hexadecimal and binary.



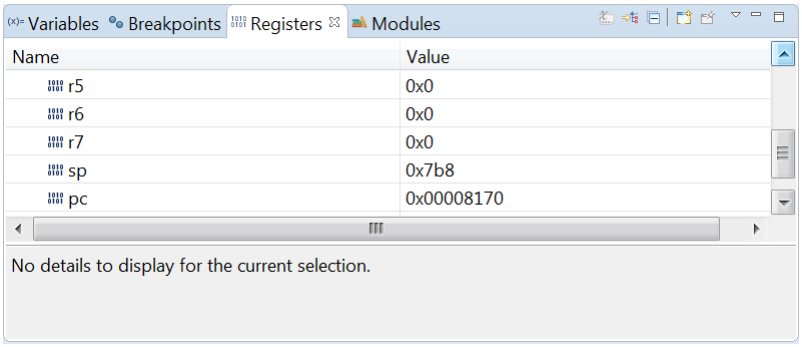
If the running program alters the register value, the display will be updated to the latest value after the program execution is suspended. The register values can be altered by clicking on the value displayed.

[Variables] view



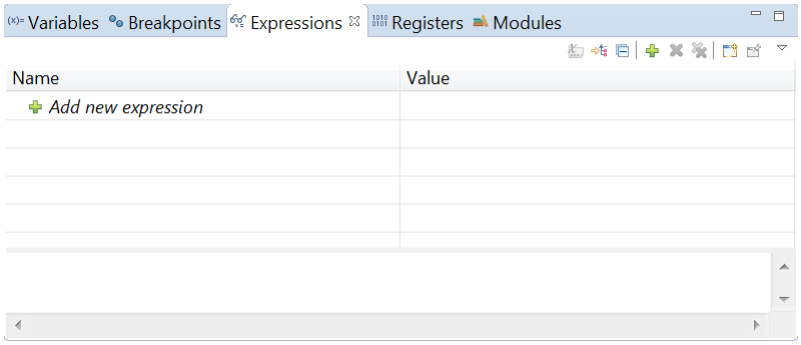
This view displays the names and values of the variables located within the scope range from the position where the target program has been suspended. The variable values can also be edited by selecting the value in the [Value] column and entering a numerical value from the keyboard.

[Registers] view



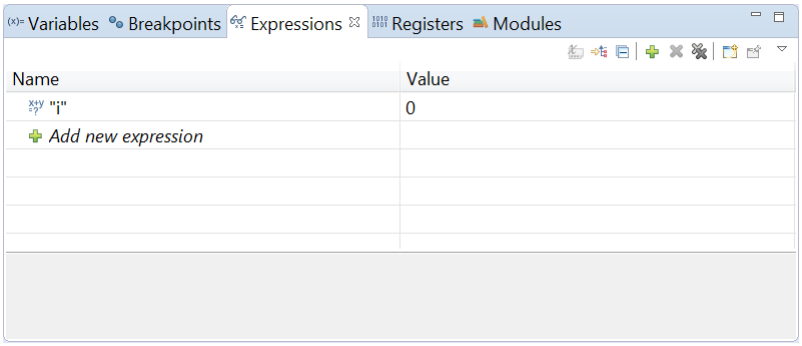
This view displays the CPU register values at the time the target program has been suspended.
The register values can also be edited by selecting the value in the [Value] column and entering a numerical value from the keyboard.

[Expressions] view



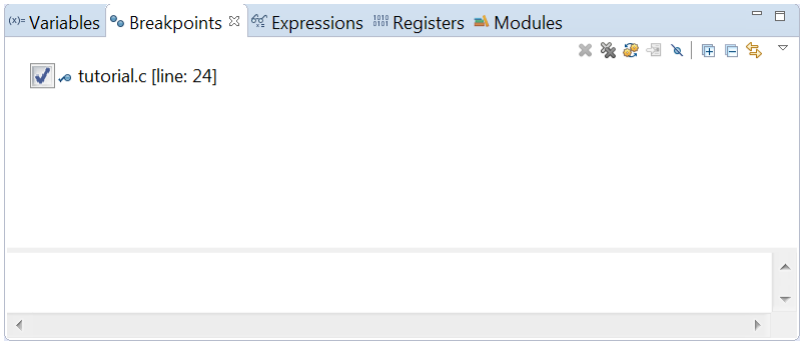
Variables can be registered to this view to monitor the values when the target program has been suspended.

Step 19: To register an expression, click [+ Add new expression] and enter the variable name to be monitored.



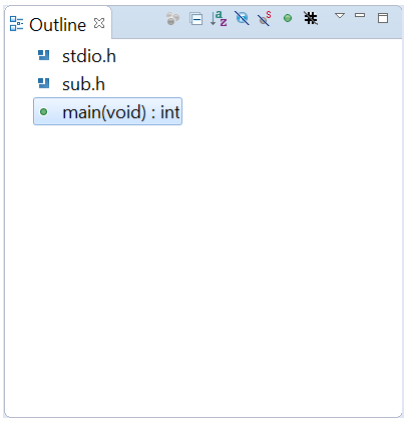
Note: When the target program moves outside the scope range of the variable, the value cannot be displayed as an error.

[Breakpoints] view



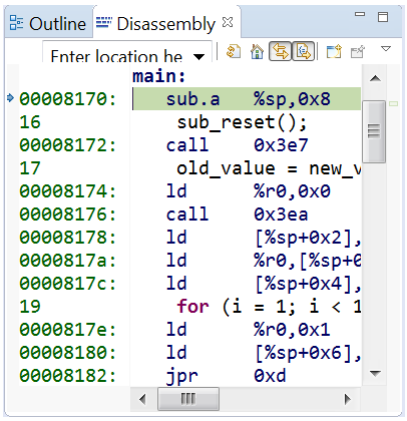
This view displays the breakpoints that have been set. Deselecting the checkbox disables the breakpoint (the program execution will not be suspended at the breakpoint until it is selected to be enabled again). For how to set breakpoints, refer to the next section.

[Outline] view



This view displays the structure (include files and functions) of the file being currently displayed in the editor. By clicking a displayed element, the editor display can be moved to the desired position quickly.

[Disassembly] view



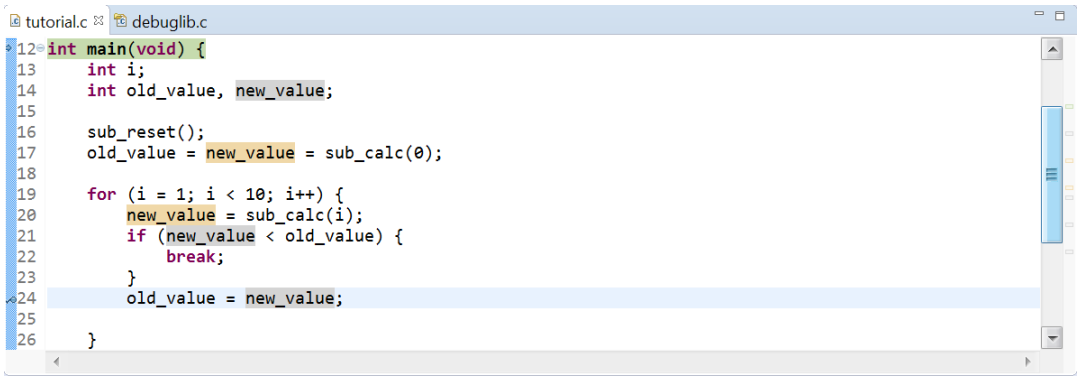
This view displays the disassembled codes of the C source from the currently suspended position.
The arrow in the marker bar at the left side of the view indicates the current line pointed by the program counter and indicates that the line has been set to be a breakpoint.
This view allows setting of breakpoints and step execution.

2.7.7 Specifying Breakpoints

The program execution can be suspended at the specified position to monitor variable and register values at that point. To do this, add breakpoints as shown below.

To add a breakpoint

Step 20: Double-click on the line number “24” of “tutorial.c” in the editor or on the marker bar at the left side of the line.

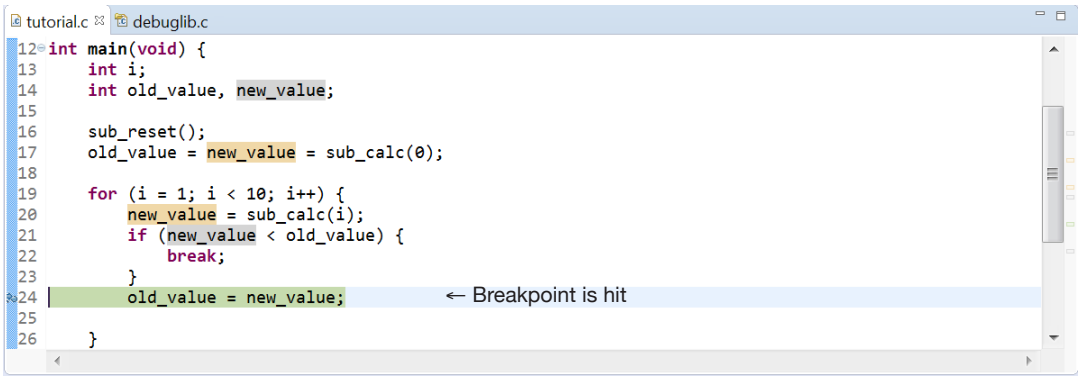


appears on the marker bar to indicate that the line is set to be a breakpoint.
Let’s execute the program again.

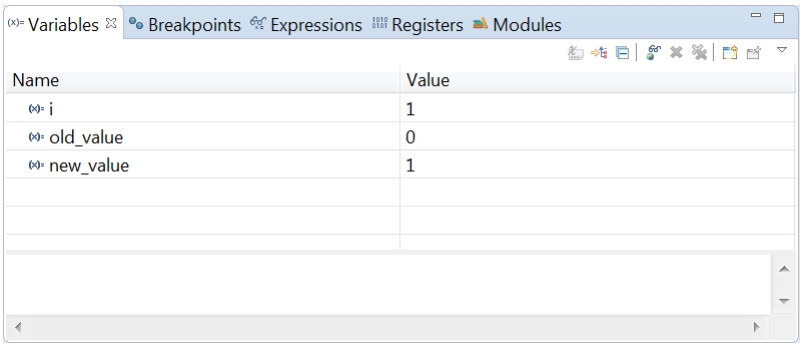
Step 21: Click the (Debug) button in the tool bar to execute from the beginning of the program.

Step 22: Click the (Resume) button in the tool bar (or select [Resume] from the [Run] menu).

The sample program starts running and stops at line 24 in “tutorial.c.”



Take a look at the [Variables] view. It shows that the variables are set as follows:



The sub_calc function calculates the 1st (i = 1) square pyramidal number and the “new_value” (result) is set to 1. This value has not been substituted to “old_value” yet, so the program is suspended before executing line 24.

Step 23: Repeat the program execution using the (Resume) button.

The variable values in the [Variables] view change as follows. It shows that the calculation is being performed correctly.

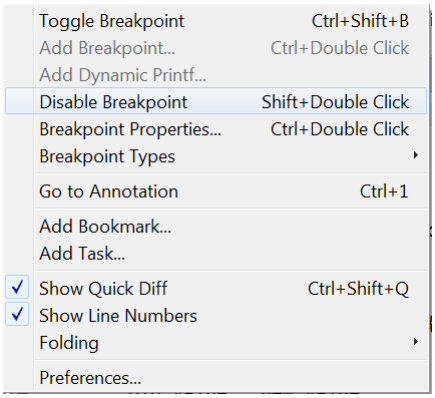
i = 1, new_value = 1 (1st square pyramidal number = 1)
i = 2, new_value = 5 (2nd square pyramidal number = 5)
i = 3, new_value = 14 (3rd square pyramidal number = 14)
: :
i = 9, new_value = 285 (9th square pyramidal number = 285)

To disable breakpoints

The following shows how to disable a breakpoint without clearing the setting.

How to operate in the editor

Step 24: Right-click on the line number or and select [Disable Breakpoint] from the context menu appeared.

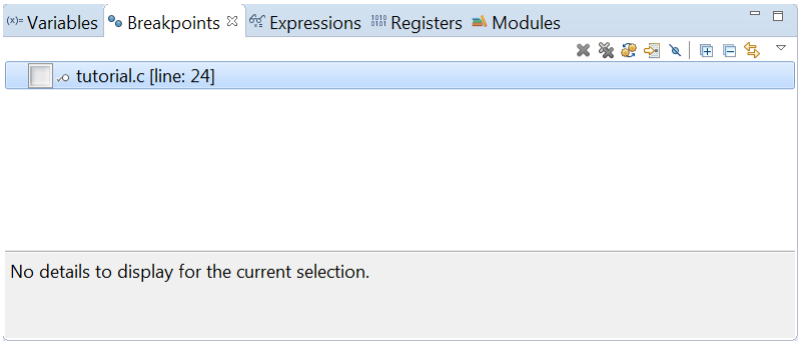


There is a limitation in the number of breakpoints that can be enabled simultaneously on the ROM. Therefore, unnecessary breakpoints should be disabled.
When the breakpoint is disabled, the marker in the editor changes to .

* To enable the breakpoint again, select [Enable Breakpoint] from the context menu that appears with the same operation.

How to operate in the [Breakpoints] view

Step 25: Deselect the checkbox of “tutorial.c [Line: 24].”



* To enable the breakpoint again, select the checkbox.

To clear breakpoints

How to operate in the editor

Step 26: Double-click on the line number “24” or (○) of “tutorial.c” in the editor.

How to operate in the [Breakpoints] view

Step 27: Select “tutorial.c [Line: 24]” and click the (Remove Selected Breakpoints) button.

∗ The breakpoint operations in the editor can also be performed in the [Disassembly] view in the same manner.

2.7.8 Step Execution

So far, the program was executed continuously. In addition, it can be executed step-by-step to check behavior of each program step.

Step 28: Click the (Debug) button in the tool bar to execute from the beginning of the program.

To execute C source lines step-by-step

To step through the source lines in a function called

Step 29: Click the (Step Into) button in the tool bar.

Every time the button is clicked, the source line highlighted in green is executed and the highlighting moves to the next source line to be executed.

1.

tutorial.c | sub.c

```
 3  Name        : tutorial.c
 8
 9  #include <stdio.h>
10  #include <sub.h>
11
12  int main(void) {
13      int i;
14      int old_value, new_value;
15
16      sub_reset();
17      old_value = new_value = sub_calc(0);
18
19      for (i = 1; i < 10; i++) {
20          new_value = sub_calc(i);
21          if (new_value < old_value) {
```

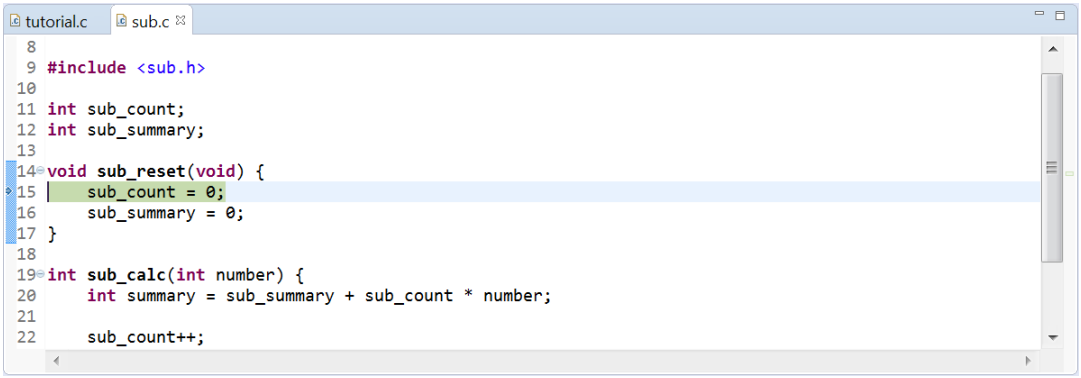
2.

tutorial.c | sub.c

```
 3  Name        : tutorial.c
 8
 9  #include <stdio.h>
10  #include <sub.h>
11
12  int main(void) {
13      int i;
14      int old_value, new_value;
15
16      sub_reset();
17      old_value = new_value = sub_calc(0);
18
19      for (i = 1; i < 10; i++) {
20          new_value = sub_calc(i);
21          if (new_value < old_value) {
```

[Step Into] executes the source lines in the sub_reset and sub_calc functions step-by-step when they are called.

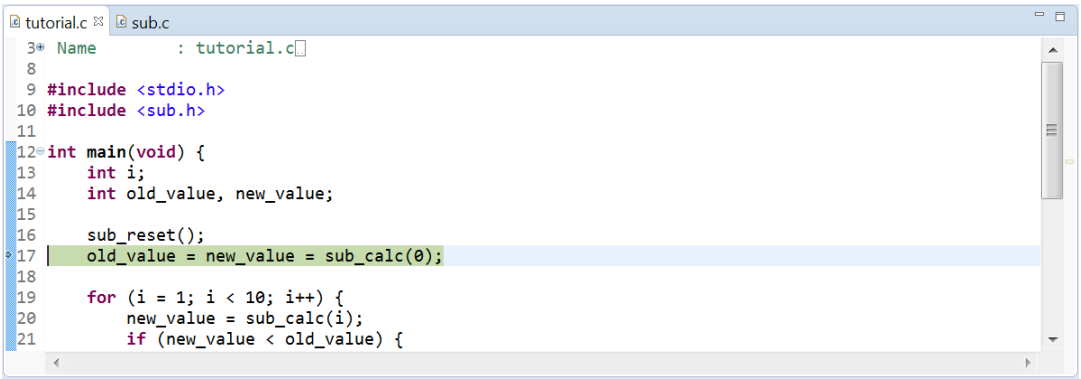
3.



Step 30: Click the (Step Return) button within a function.

[Step Return] returns from within a function to the caller in one step by continuously executing the remainder of the function being currently stepped through. The (Step Return) button will be available when a function is called.

4.

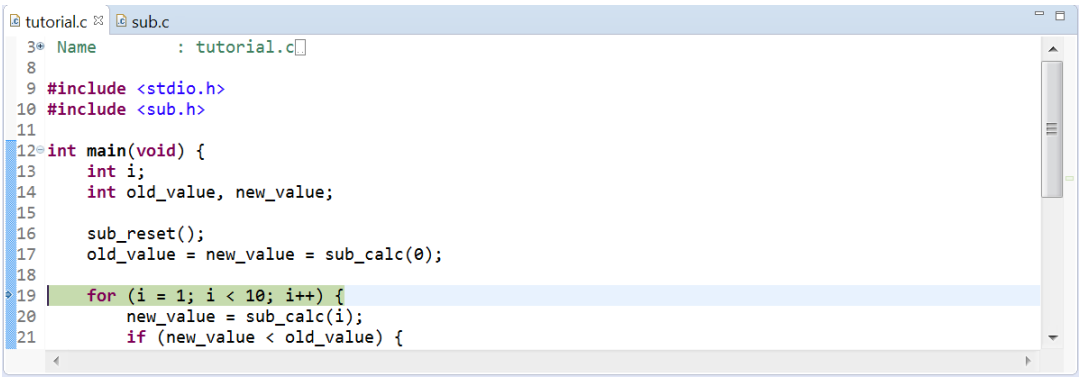


To continuously execute the source lines in a function

Step 31: Click the (Step Over) button in the tool bar.

As with [Step Into], [Step Over] steps through one line at a time. However, functions are executed continuously as one line from being called to return. Thus, the source lines within the functions that do not need debugging can be skipped.

5.



To step through the program in mnemonic instruction units

Step 32: Click the (Instruction Stepping Mode) button to turn it to ON.

The [Disassembly] view becomes active and the subsequent step execution will be performed in mnemonic instruction units within the [Disassembly] view.
To return to the C source step execution, click the (Instruction Stepping Mode) button to turn it to OFF.
The operations to step through are the same as the C source described above.

[Step Into]	Executes one instruction at a time including the instructions within subroutines called.
[Step Over]	Executes one instruction at a time but subroutines called are executed continuously.
[Step Return]	Returns from a subroutine to the caller in one step by continuously executing the remaining instructions in the subroutine.

2.7.9 Reset

Resets can be issued to the target CPU or target board by selecting a menu item.

To reset the target CPU

Step 33: Select [Reset] from the [c17] menu.

The debugger executes the GDB command file "reset.gdb" to reset the target CPU.

To reset the target board

Step 34: Select [Reset Target] from the [c17] menu.

The debugger executes the GDB command file "reset_target.gdb" to reset the target board. This function is effective only in ICDmini mode. Furthermore, the target board must be equip with a pin for inputting a reset signal from the ICDmini.

2.7.10 C17-Specific Debug Functions

The [C17] menu provides the following functions in addition to the reset issuance functions shown in the previous section:

1. Launch LcdUtility

Launches the "LcdUtil17" utility used to design the LCD panel screen for the peripheral circuit simulator. For detailed information on the "LcdUtil17," refer to Section 10.8, "LCDUtil17 (LCD Panel Customizing Tool)," in the "S5U1C17001C Manual."

2. User Command

Executes the GDB command file "userdefine.gdb." This file is generated in the project folder when the project is newly created. The user can edit the contents of this file freely using the editor.

3. Debug Command

The following commands can be executed by selecting from the sub-menu:

c17 rst	Resets the CPU.
c17 rstt	Outputs a reset signal to the target board. (ICDmini mode only)
c17 int	Issues an interrupt request of the specified interrupt number. (Simulator mode only)
c17 intclear	Cancels the interrupt request of the specified interrupt number. (Simulator mode only)
c17 tm	Configures the trace function. (Simulator mode only)
c17 chgclkmd	Configures whether DCLK is switched to the high-speed clock when a break occurs or not. (ICDmini mode only)
c17 flv	Sets the flash programming voltage of ICDmini Ver. 2.0. (ICDmini mode only)
c17 flvs	Clears the flash programming voltage set to ICDmini Ver. 2.0. (ICDmini mode only)

For details of each command, refer to Section 8.5, `Command Reference," in the "S5U1C17001C Manual."

2.7.11 Terminating the Debugger

Step 35: Click the  (Terminate) button in the tool bar.

To switch the perspective back to [C/C++], click the [C/C++] button. To terminate the IDE, select [Exit] from the [File] menu.

3 Tutorial 2 (Importing an Existing Project)

If an S1C17 Family application has been developed using the IDE, the development can be continued or the program can be revised by importing that project into the IDE on another PC. Another application can also be developed based on that project. This chapter shows how to import projects. For other procedures, refer to Tutorial 1.

3.1 Importing a GNU17 Ver. 3.x project

The projects created using GNU17 Ver. 3.x can be imported without any modification.

Sample project folder used

C:\EPSON\GNU17V3\sample\tutorial

The project folder in the workspace cannot be overwritten by importing a project with the same name using the IDE. Therefore, the tutorial project created in Tutorial 1 must be deleted before starting this tutorial.

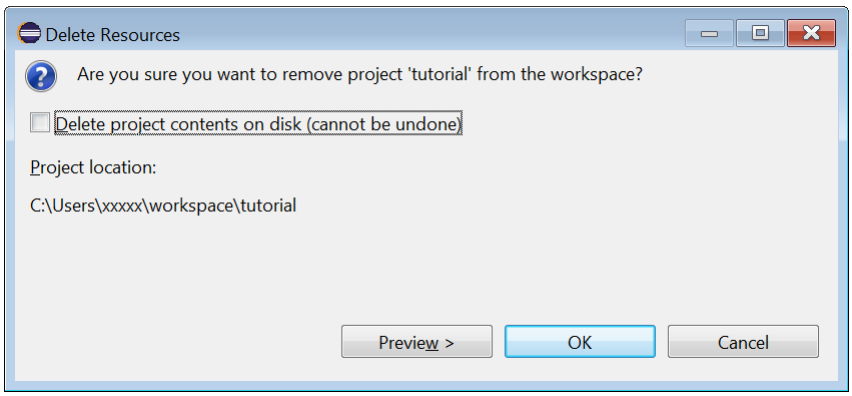
Step 1: Launch the IDE.

If Tutorial 1 has not been started yet, the following operations (Steps 2 and 3) to delete the project are not necessary.

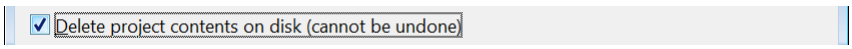
To delete a project

Step 2: Select “tutorial” in the [Project Explorer] view and press the [Delete] key or select [Delete] from the [Edit] menu (or context menu that appears by right-clicking).

This will bring up the [Delete Resources] dialog box.



Step 3: Select the [Delete project contents on disk (cannot be undone)] checkbox and click the [OK] button.

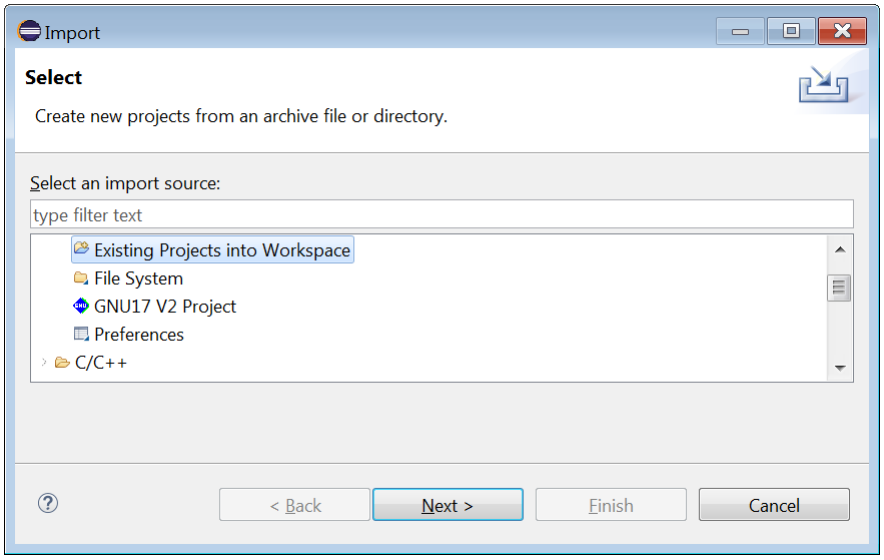


* The [Delete project contents on disk (cannot be undone)] checkbox is deselected by default. Clicking the [OK] button with the checkbox deselected deletes the tutorial project from the [Project Explorer] view but the project folder on the disk is not deleted. It can be imported again.
  Clicking the [OK] button with the checkbox selected deletes the project from the workspace directory on the disk as well as from the [Project Explorer] view. It is no longer able to be bring back.

To import a GNU17 Ver. 3.x project

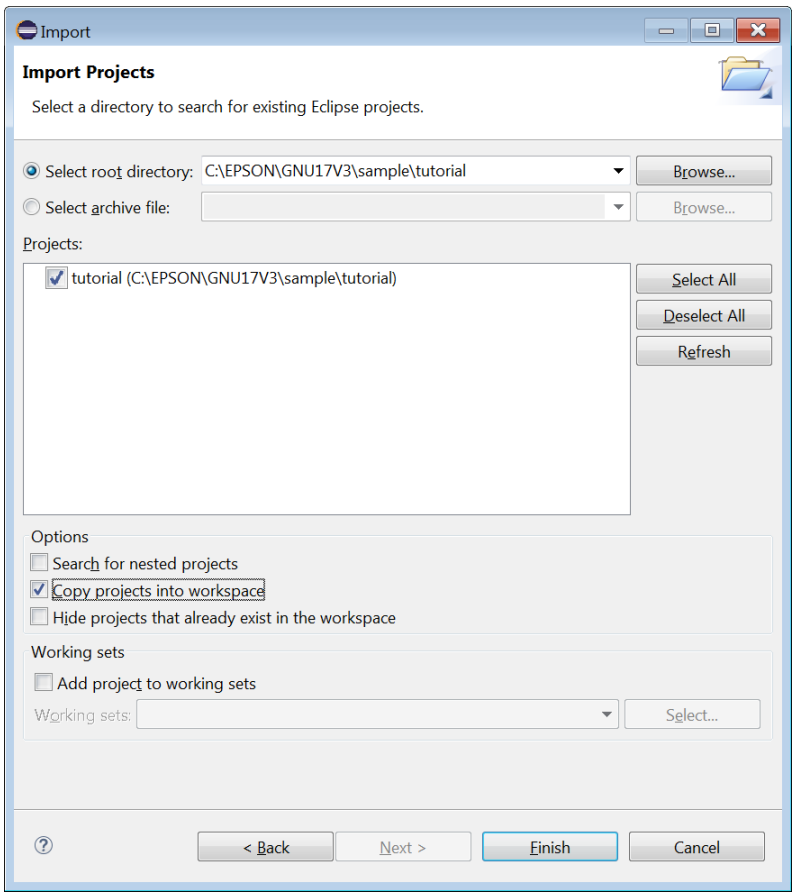
Step 4: Launch the IDE and select [Import...] from the [File] menu.

The [Import] wizard will start.



Step 5: Select [General] > [Existing Projects into Workspace] from the list being displayed and click the [Next >] button.

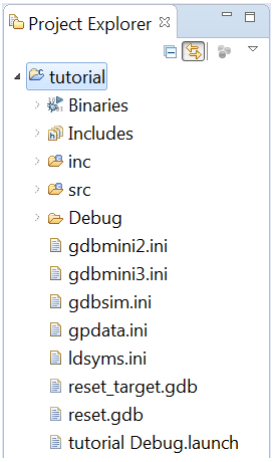
Step 6: Select the project folder to be imported, “C:\EPSON\GNU17V3\sample\tutorial,” using the [Browse...] button at [Select root directory:].



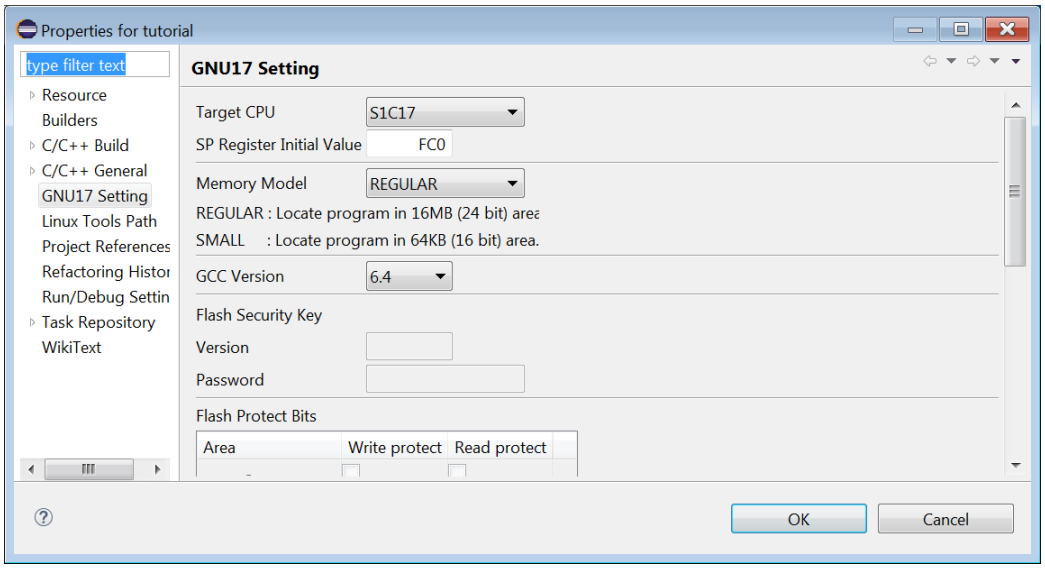
Selecting the [Copy projects into workspace] check box will make a copy of the project into the workspace directory and the original project files will not be modified.

Step 7: Click the [Finish] button.

The imported project appears in the [Project Explorer] view.



Step 8: Select “tutorial” in the [Project Explorer] view and select [Properties] from the [Project] menu (or context menu that appears by right-clicking) to bring up the [Properties] dialog box. Then, select [GNU17 Setting] from the property list.



Confirm the basic configuration of the project in this page and reconfigure if necessary.

Changing the GCC version

Step 9: Select the version of the C compiler “gcc” to be used from the [GCC Version] drop-down list as necessary. Normally use with [6.4].

* When the GCC version is changed, check if the settings shown below are updated in the [Properties] dialog box.

1) [C/C++ Build] > [Environment] > [GCC17-LOC] > [Value]

<When 4.9 is used>	<When 6.4 is used>
${GNU17_LOC}\gcc4	${GNU17_LOC}\gcc6

2) [C/C++ Build] > [Settings] > [Tool Settings] > [Cross Settings] > [Path]

<When 4.9 is used>	<When 6.4 is used>
C:\EPSON\GNU17V3\gcc4	C:\EPSON\GNU17V3\gcc6

3) [C/C++ Build] > [Settings] > Tool Settings] > [Cross GCC Compiler] > [Optimization] > [Optimization Level]

<When 4.9 is used>	<When 6.4 is used>
-O0/-O1/-O2/-O3/-Os	-O0/-O1/-Os

4) [C/C++ Build] > [Settings] > [Tool Settings] > [Cross GCC Compiler] > [Debugging] > [Other debugging flags]

<When 4.9 is used>	<When 6.4 is used>
-gstabs	-g

3.2 Importing a GNU17 Ver. 2.x Project

This section shows a procedure to import a project created using GNU17 Ver. 2.x into GNU17 Ver. 3.x and to build it. The startup processing library "crt0.o," which is a GNU17 Ver. 3.x function, is not used. The following operation procedure is required.

1. Preparation of a linker script file
2. Configuration of linker options
3. Modification of the source code
4. Modification of the GDB command file

The following shows this operation procedure.

Preparing the sample project

The sample project used in this tutorial is provided as a ZIP file in the "C:/EPSON/GNU17V3/sample" folder. First, extract this file.

Step 1: Right-click "tutorial_v2import.zip" in the sample folder to bring up a context menu and select [Extract All...] to extract the file. In the dialog box that appears, specify the folder shown below as the destination.

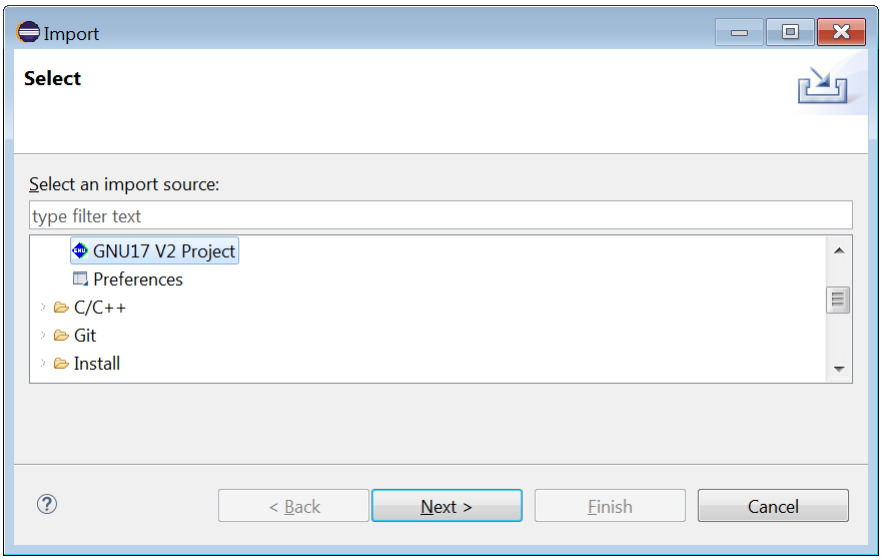
C:\EPSON\GNU17V3\sample\tutorial_v2import

If this project has already been imported, delete the "tutorial_v2import" project including the project folder on the disk (refer to "To delete a project" in Section 3.1).

To import a GNU17 Ver. 2.x project

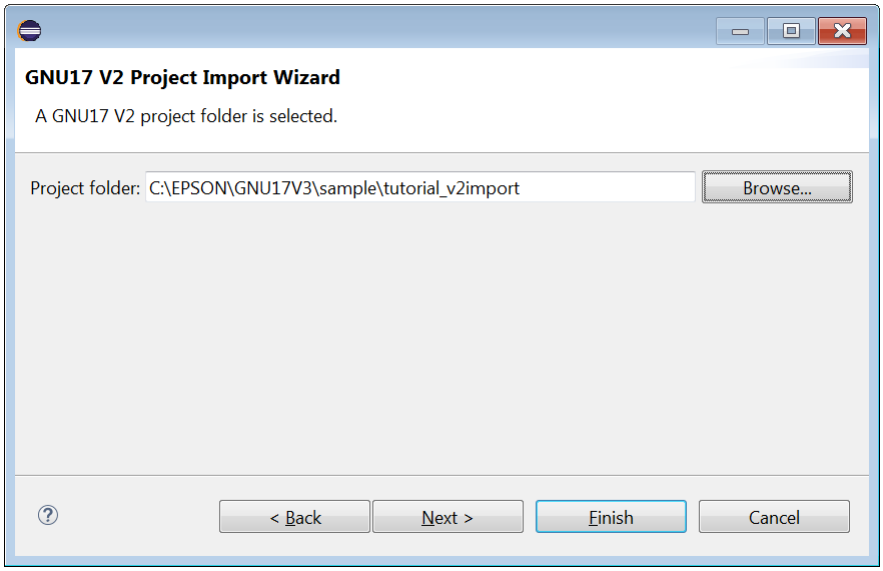
Step 2: Launch the IDE and select [Import...] from the [File] menu.

The [Import] wizard will start.



Step 3: Select [General] > [GNU17 V2 Project] from the list being displayed and click the [Next >] button.

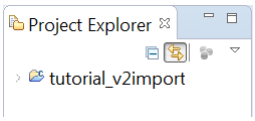
Step 4: Select the project folder to be imported, "C:\EPSON\GNU17V3\sample\tutorial_v2import," using the [Browse...] button at [Project folder:].



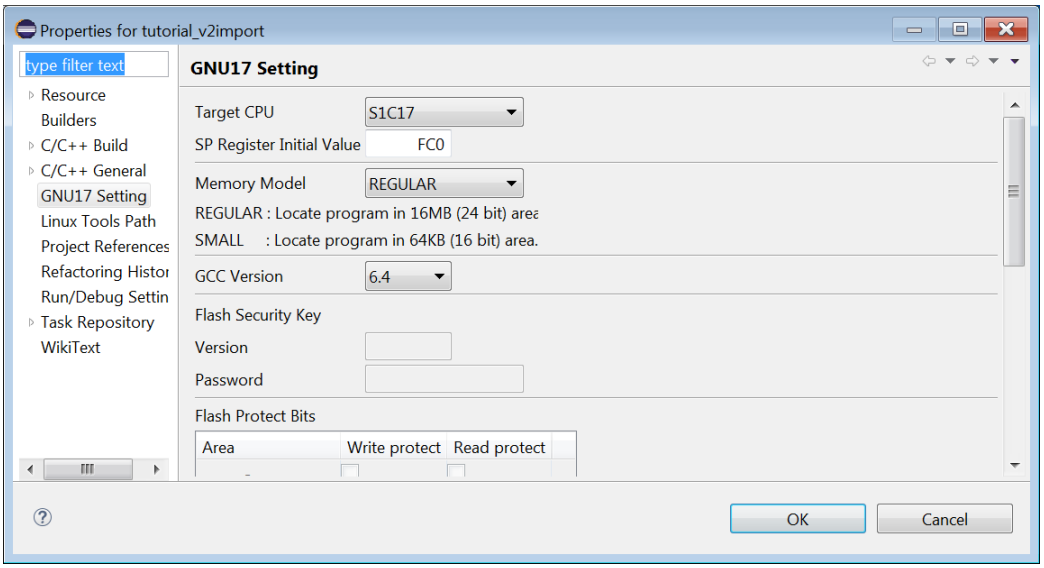
Step 5: Click the [Finish] button.

Step 6: “ImportV2Note” (C:\EPSON\GNU17V3\doc\ImportV2Note.txt) is displayed. Close it after reading.

The imported project appears in the [Project Explorer] view.



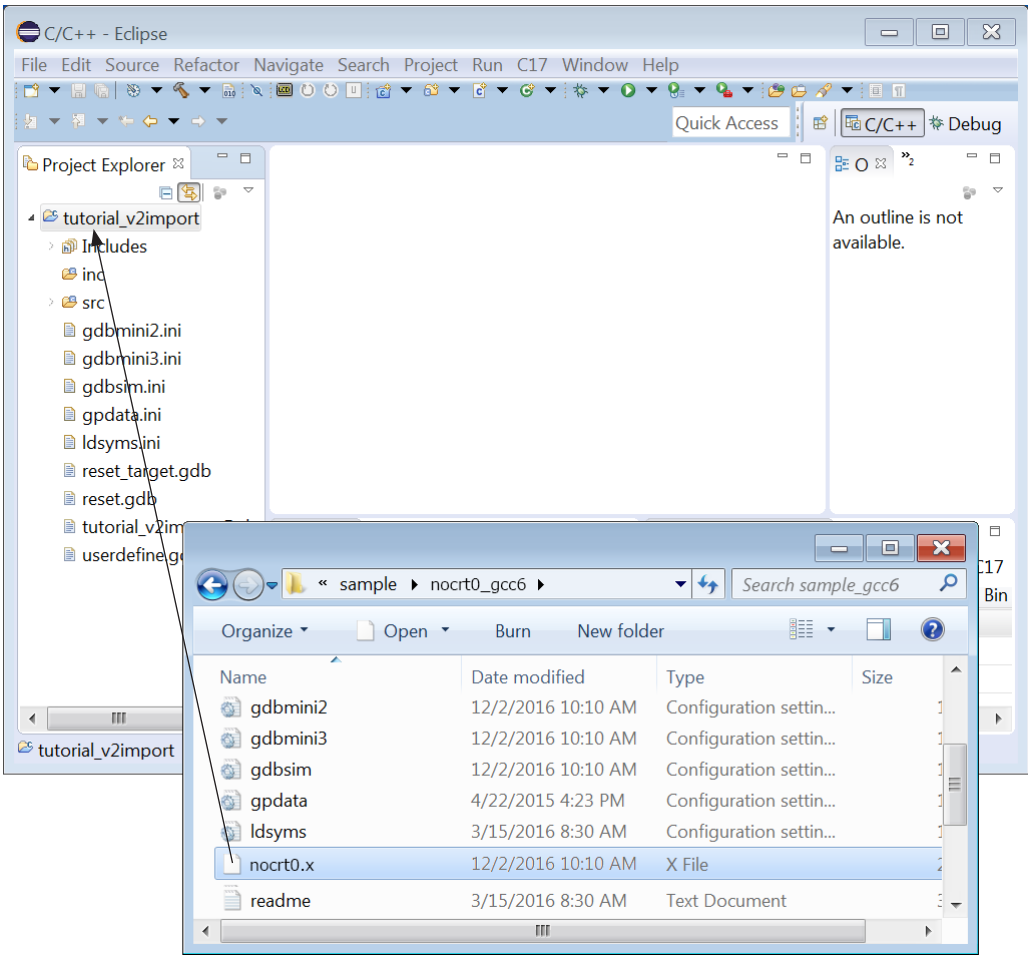
Step 7: Select “tutorial_v2import” in the [Project Explorer] view and select [Properties] from the [Project] menu (or context menu that appears by right-clicking) to bring up the [Properties] dialog box. Then, select [GNU17 Setting] from the property list.



Confirm the basic configuration of the project in this page and reconfigure if necessary.

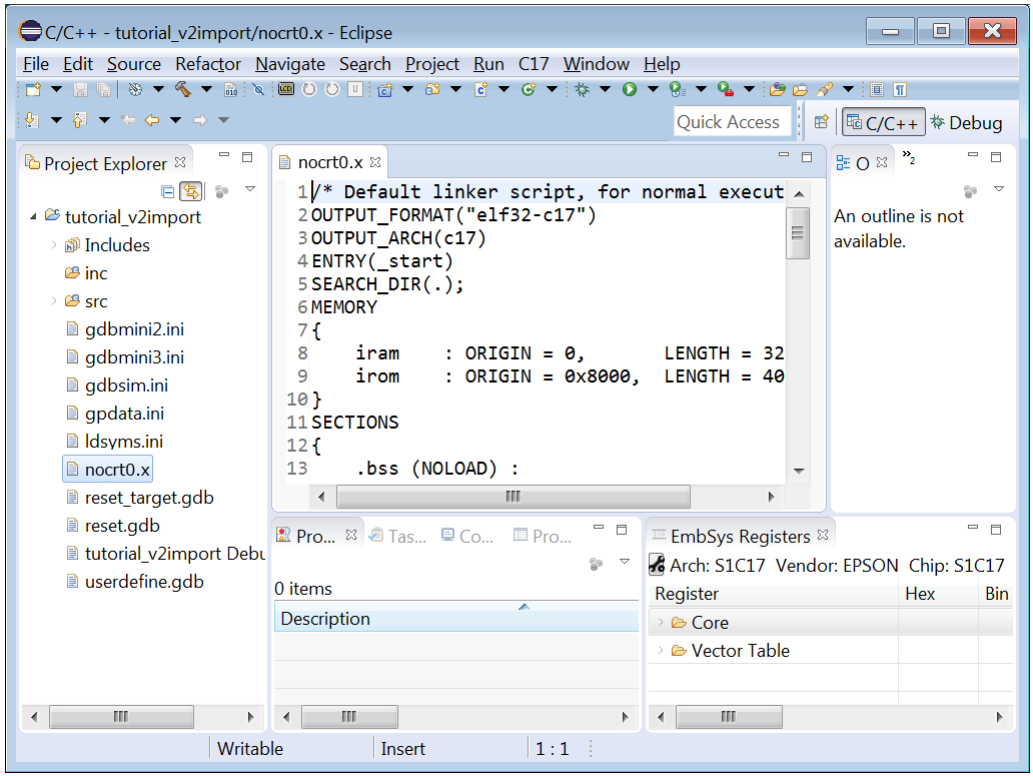
Preparing a linker script file

Step 8: Drag and drop the “nocrt0.x” file on the “tutorial_v2import” project in the [Project Explorer] view from the “C:\EPSON\GNU17V3\sample\nocrt0_gcc6” folder. When the [File Operation] dialog box appears, select [Copy files] and click the [OK] button.



Edit this linker script file before being used.
Modify the specification of the object file that defines the vector table; replace “boot.o” with the file created by the user (“vector.o” in this sample).

Step 9: Double-click “nocrt0.x” in the [Project Explorer] view to open it with the editor.



Step 10: Correct the following three parts and save the file.

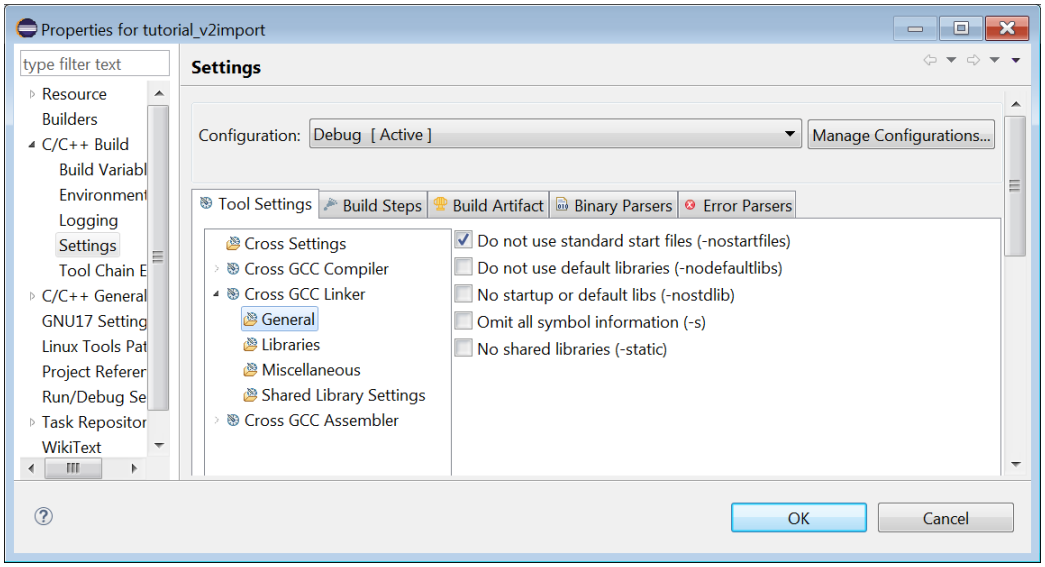
1. Change the boot processing routine.
   `ENTRY(boot)` → `ENTRY(boot)`
   * It is not necessary to edit this part in this example.
2. Change the vector table.
```
.vector :
{
        PROVIDE (__START_vector = .) ;
        KEEP (*boot.o(.rodata))          → KEEP (*vector.o(.rodata))
}  > irom
```
3. Exclude the vector table from the ".rodata" section.
```
.rodata :
{
        PROVIDE (__START_rodata = .) ;
        *(EXCLUDE_FILE (*boot.o) .rodata) → *(EXCLUDE_FILE (*vector.o) .rodata)
        *(.rodata.*)
        PROVIDE (__END_rodata = .) ;
}  > irom
```

Configuring linker options

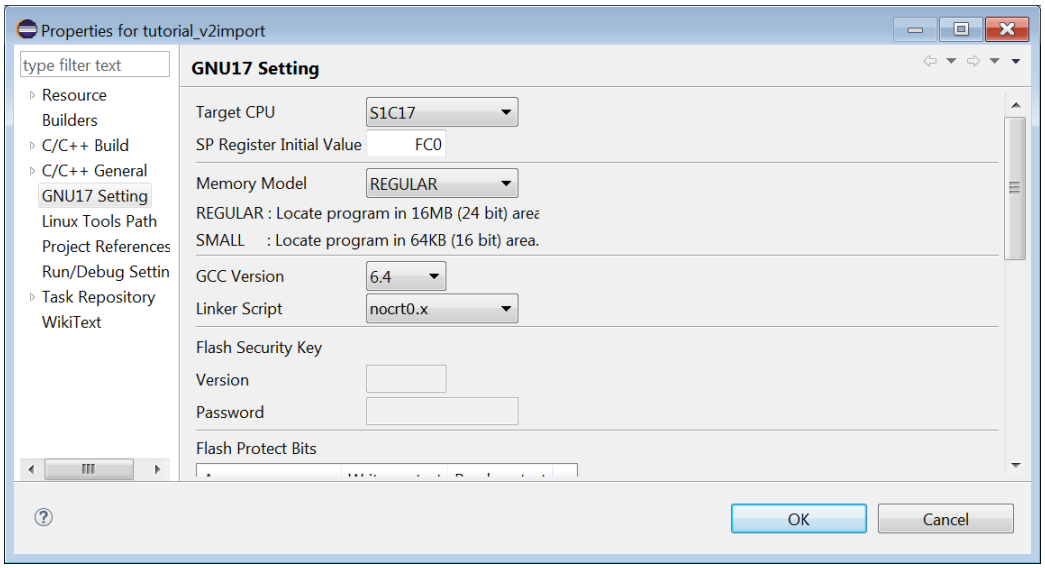
Configure the following linker options so that the linker script file prepared will be used.

Step 11: Select "tutorial_v2import" in the [Project Explorer] view and select [Properties] from the [Project] menu (or context menu that appears by right-clicking) to open the [Properties] dialog box. Then select [C/C++ Build] > [Setting] to open the [Tool Settings] tab page.

Step 12: Select [Cross GCC Linker] > [General] from the setting list in the [Tool Settings] page and select [Do not use standard start files (-nostartfiles)] in the page that appears.



Step 13: Open the [GNU17 Setting] page of the [Properties] dialog box and select “nocrt0.x” from the [Linker Script] drop-down list.



Step 14: Click the [OK] button to close the [Properties] dialog box.

Editing the source code

If the boot processing (“void boot(void)” in “vector.c” in this example) or another routine has the following descriptions, the source code should be edited for using the standard I/O and other functions.

1. **_init_sys();**
   Delete “_init_sys();” because it has not been defined in GNU17 Ver. 3.x.

2. **_init_lib();**
   Add (1) and (2) shown below because “_init_lib();” has not been defined in GNU17 Ver. 3.x.

   (1) Delete “#include <libstdio.h>” and add the following descriptions:

```
#include <smcvals.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

(2) Add a "_init_lib" function.

```
static void _init_lib(void);
…
int errno;
FILE _iob[FOPEN_MAX + 1];
FILE *stdin;
FILE *stdout;
FILE *stderr;
unsigned long seed;
time_t gm_sec;
…
static void _init_lib(void) {
    /* initalize for general */
    errno = 0;                  /* clear error number */

    /* initalize for io stream in io.lib */
    _iob[0]._flg = _UGETN;      /* initialize stdin stream */
    _iob[0]._buf = 0;
    _iob[0]._fd = 0;

    _iob[1]._flg = _UGETN;      /* initialize stdout stream */
    _iob[1]._buf = 0;
    _iob[1]._fd = 1;

    _iob[2]._flg = _UGETN;      /* initialize stderr stream */
    _iob[2]._buf = 0;
    _iob[2]._fd = 2;

    stdin = &_iob[0];           /* initialize each file pointer */
    stdout = &_iob[1];
    stderr = &_iob[2];

    /* initialize for others in lib.lib */
    seed = 1;                   /* initialize random seed */
    gm_sec = -1;                /* initialize time */
}
```

3. _exit();

Edit the (1) to (3) parts shown below because "_exit();" has been defined in GNU17 Ver. 3.x.

(1) Delete "void _exit(void);" and add the following description:

```
extern void _exit(int);
```

(2) Delete "void _exit(void);."

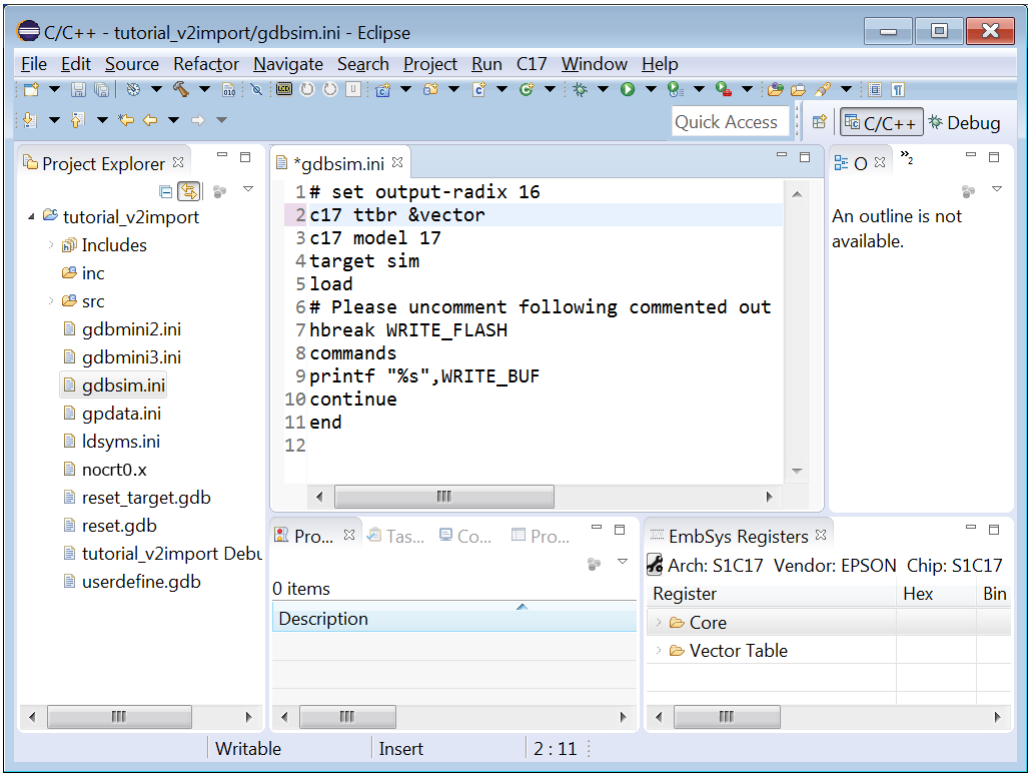
(3) Add an argument to "_exit();."

```
void boot(void)
{
   …
   _exit(0);                    // In last, go to exit in sys.c
}
```

Editing the GDB command file

Correct the specified vector table address to the user definition ("vector" in this sample).

Step 15: Double-click "gdbsim.ini" in the [Project Explorer] view to open it with the editor and change "c17 ttbr &_vector" to "c17 ttbr &vector." Then save the file.



The necessary modification has been completed. Hereinafter, building and debugging the project can be performed in the normal operation procedure.

Appendix A Sections and Linker Script

A.1 Sections

This section describes the concept of section management that is required when creating source files and linking.
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 arranged in different memory devices such as ROM (flash memory) 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 codes present in multiple source files will be arranged in one section, program codes can easily be combined from these source files when linked, and will consequently be arranged in the same ROM. And since addresses can be specified separately for each file, they can be arranged on separate devices, such as internal ROM and external ROM.
Four broad categories (attributes) of sections are supported by the C compiler xgcc, and data is arranged in the appropriate section according to the contents of the source files.

(1) .text section

Program codes are arranged here. All codes are eventually written to ROM.

(2) .data section

Readable/writable data with initial values are arranged here. The initial values are written to ROM, from which it is transferred to RAM by the program before use.

(3) .rodata section

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

Although the vector table has the ".rodata" attribute, the dedicated output section ".vector" is provided, as it must be arranged in the specific location. In GNU17 Ver. 3.x, the boot processing library "crt0.o," which will be linked by default, includes a vector table and it will be arranged in the ".vector" section.

(4) .bss section

Variables without initial values are arranged here. A RAM space is allocated without a specific value.

A.2 Linker Script

The linker script is used to specify storing and executing locations of each section, and to control linking. A default linker script is incorporated in the linker and is used unless otherwise specified. A custom linker script file created by the user can also be used.
The following shows an example of standard linker script. To display the default linker script, execute "ld --verbose" at a command prompt.

Standard linker script example

```
OUTPUT_FORMAT("elf32-c17")                                          (1)
OUTPUT_ARCH(c17)
ENTRY(_start)
SEARCH_DIR(.);
MEMORY                                                              (2)
{
     iram : ORIGIN = 0, LENGTH = 32K
     irom : ORIGIN = 0x8000, LENGTH = 4064K
}
SECTIONS                                                            (3)
{
     .bss (NOLOAD) :
     {
          PROVIDE (__START_bss = .) ;
          *(.bss)
          *(.bss.*)
          *(COMMON)
          PROVIDE (__END_bss = .) ;
     } > iram
     .vector :
     {
          PROVIDE (__START_vector = .) ;
          KEEP (*crt0.o(.rodata))
          PROVIDE (__END_vector = .) ;
     } > irom
     .text :
     {
          PROVIDE (__START_text = .) ;
          *(.text.*)
          *(.text)
          PROVIDE (__END_text = .) ;
     } > irom
     .data :
     {
          PROVIDE (__START_data = .) ;
          *(.data)
          *(.data.*)
          PROVIDE (__END_data = .) ;
     } > iram AT > irom
     .rodata :
     {
          PROVIDE (__START_rodata = .) ;
          *(EXCLUDE_FILE (*crt0.o ) .rodata)
          *(.rodata.*)
          PROVIDE (__END_rodata = .) ;
     } > irom
     PROVIDE (__START_data_lma = LOADADDR(.data));
     PROVIDE (__END_data_lma = LOADADDR(.data) + SIZEOF (.data));
     PROVIDE (__START_stack = 0x0007C0);
}
```

(1) OUTPUT_FORMAT ... SEARCH_DIR

OUTPUT_FORMAT(“elf32-c17”)	Output file format
OUTPUT_ARCH(c17)	Target architecture
ENTRY(_start)	Entry point (program start address) “_start” points the address of the boot processing function in the boot processing library “crt0.o.” If “_start” is specified in the linker command line, it takes precedence.
SEARCH_DIR(.);	Library search path If a path is specified in the linker command line, it takes precedence.

This part is fixed. Write the same contents when creating a custom linker script file.

(2) MEMORY

The target memory configuration is defined. “iram” and “irom” are the area names that represent the internal RAM and internal ROM (flash memory), respectively. In GNU17, attribute descriptions are omitted. “ORIGIN” specifies the top address and “LENGTH” specifies the capacity.
The default linker script specifies a memory configuration applicable to many S1C17 microcontrollers whose internal RAM is located from address 0 and the internal ROM is located from address 0x8000.
When creating a custom linker script file, write the memory configuration of the target CPU here. Names other then “iram” and “irom” can be used according to the target memory configuration.

(3) SECTIONS

Sections are defined in the arrangement order from the top of memory.
The basic definition format is as follows:

Output section name [**(***Section type***)**] **:**
{
  PROVIDE (*Symbol name* **= .) ;**
  (Input section name***)**
    :
  PROVIDE (*Symbol name* **= .) ;**
} > *VMA region name* [**AT >** *LMA region name*] [ ] can be omitted.

This definition allocates memory “*VMA region name*” for “*Output section*” and arranges data within the “*Input section*,” which have been defined in the object files to be linked, to the “*Output section*” in the order of files specified.

’*’ in “*(*Input section name*)” is a wildcard character that specifies all the “*Input section*” included in the object files to be linked.

“AT > *LMA region name*” specifies a data storage region different from the execution region. For example, this definition is used for “.data” section whose initial data is stored in a ROM (*LMA region*) and is copied to a RAM (*VMA region*) for accessing at execution time. VMA and LMA mean “Virtual Memory Address” and “Load Memory Address,” respectively. When “AT > *LMA region name*” is omitted, it is regarded as “*VMA region*” = “*LMA region*.”

“PROVIDE (*Symbol name* = .) ;” defines a symbol “*Symbol name*” as ‘.’ (location counter value = address of the described position). For example, “PROVIDE (__START_bss = .) ;” defines the symbol “__START_bss” as the internal RAM top address. “PROVIDE (__END_bss = .) ;” defines the symbol “__END_bss” as the “.bss” section end address that is decided after the input sections are all arranged. These symbols can be used in the target program. If the same symbol is defined in a source file, the definition in the source file takes effect.

The following shows other scripts used in the example above.

.bss (NOLOAD)

NOLOAD is a keyword for specifying a section type and it represents a section in which no data exists (no actual codes and data will be stored by linking).

KEEP (*crt0.o(.rodata))

KEEP is the keyword that makes certain to include the specified input section in the output section. In this example, the ".rodata" section (vector table) in "crt0.o" is output as the ".vector" section.

***(EXCLUDE_FILE (*crt0.o) .rodata)**

EXCLUDE_FILE is the keyword to not include the specified input section in the output section. In this example, the ".rodata" section in "crt0.o," which has been output to the ".vector" section, is excluded from the outputs to the ".rodata" section.

PROVIDE (__START_data_lma = LOADADDR(.data));

LOADADDR represents the storing region top address. This example defines the symbol "__START_data_lma" as the top address of the region for storing the ".data" section in the ROM.

PROVIDE (__END_data_lma = LOADADDR(.data) + SIZEOF (.data));

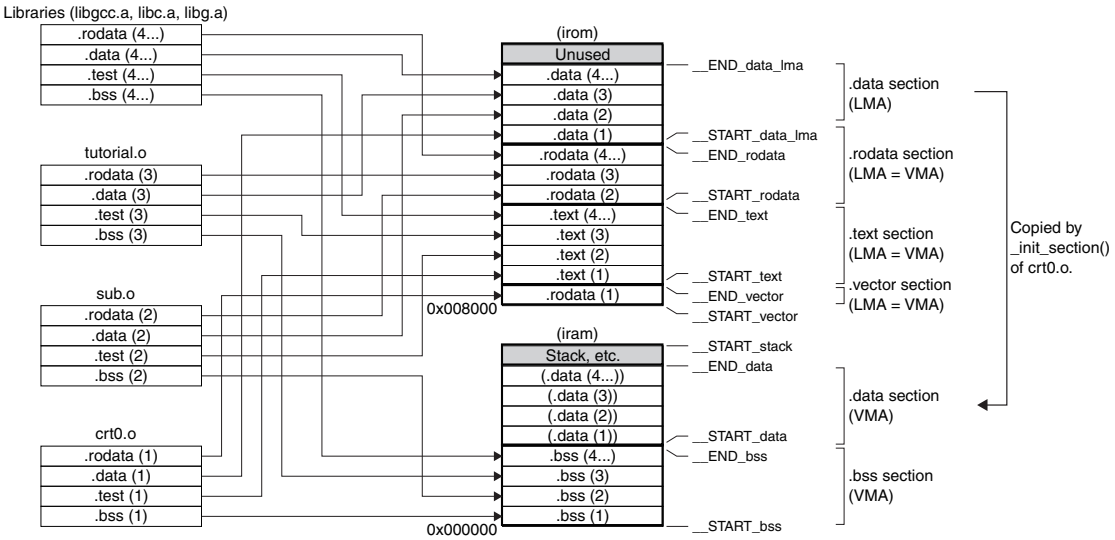
SIZEOF is the keyword to obtain the size of the specified section. This example defines the symbol "__END_data_lma" as the end address of the region for storing the ".data" section in the ROM.

PROVIDE (__START_stack = 0x0007C0);

This example defines the symbol "__START_stack" as the stack pointer initial value. If this symbol is defined in a source file as shown below, the definition in the linker script is ignored and that in the source file becomes effective.

```
asm(".global __START_stack");
asm(".set __START_stack, 0x1fc0");
```

In this linker script example, data are arranged from address 0 in the order of ".bss" and ".data." The vector table, program codes, and constant data are arranged from address 0x8000. Figure A.2.1 shows the memory map after the files used in Tutorial 1 are linked.



(This memory map contains the sections that do not exist in the objects.)

Figure A.2.1 Memory Map Configured by Linker Script Example

The program is stored from address "__START_vector" to address "__END_data_lma" in the ROM. The program uses the RAM region from address "__START_bss" to address "__END_data" and the region allocated for the stack (and heap).
The amount of ROM and RAM used by this program can be calculated as follows:

ROM usage [bytes] = __END_data_lma - __START_vector
RAM usage [bytes] = __END_data - __START_bss + stack usage (+ heap usage)

The symbol values defined in each section are output to the link map file. Refer to "tutorial\Debug\tutorial.map" as an example.

A.3 Linker Script Examples

This section shows linker script description examples. In addition to these examples, refer to the linker script files for the sample programs that are provided in the folder below.

C:\EPSON\GNU17V3\sample

These examples assume that the following conditions are satisfied.

- The sections to be added to the target project have already been defined.
- The MEMORY definition shown below has been described in the linker script file.

```
MEMORY
{
   iram  : ORIGIN = 0,          LENGTH = xxK
   irom  : ORIGIN = 0x8000,     LENGTH = xxK
}
```

Example 1) To place data on the specified address

Define data in the program.

```
const unsigned char __attribute__ ((section (".testdata"))) checkerLineBit[16] = {
   0x00, 0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F
};
```

Place the defined data from Address 0xB000.

```
.testdata (0xB000) :                    -> Specify Address 0xB000.
{
   *(.testdata);
   . = ALIGN(0x800);                    -> Set the location counter to a 2KB boundary address.
} > irom = 0xff                         -> Place the data in irom and fill the free area with 0xffff.
```

Example 2) To place the object code of the program "test.c" in the Flash and to execute it on the RAM.

```
…
.text :
{
   PROVIDE (__START_text = .) ;
   *(.text.*)
   *(EXCLUDE_FILE (*test.o ) .text)     -> Exclude "test.o" from the ".text" section.
   …
   PROVIDE (__END_text = .) ;
} > irom
…
.rodata :
{
   PROVIDE (__START_rodata = .) ;
   …
   *(EXCLUDE_FILE (*test.o) .rodata)    -> Exclude "test.o" from the ".rodata" section.
   *(.rodata.*)
   PROVIDE (__END_rodata = .) ;
} > irom
…
```

```
.test_text :                        -> Add ".test_text" section.
{
   __START_test_text = . ;
   *test.o(.text);                  -> Locate "text.o" as attribute ".text."
   __END_test_text = . ;
} > iram AT > irom                  -> Specify to place the actual code in irom and to execute it on iram.

.test_rodata :                      -> Add ".test_rodata" section.
{
   __START_test_rodata = . ;
   *test.o(.rodata);                -> Locate "text.o" as attribute ".rodata"
   __END_test_rodata = . ;
} > iram AT > irom                  -> Specify to place the actual code in irom and to execute it on iram.

__START_test_text_lma = LOADADDR(.test_text);
__START_test_rodata_lma = LOADADDR(.test_rodata);
                                    -> Specify the start address of the section added to __START_test_xxxx_lma.
...
```

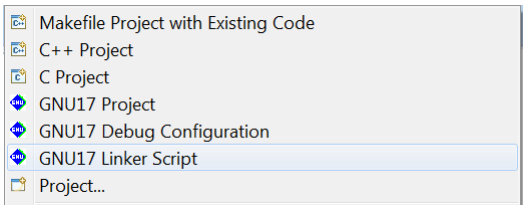
A.4 Linker Script Generation Wizard

This IDE provides a linker script generation wizard for generating a linker script file.

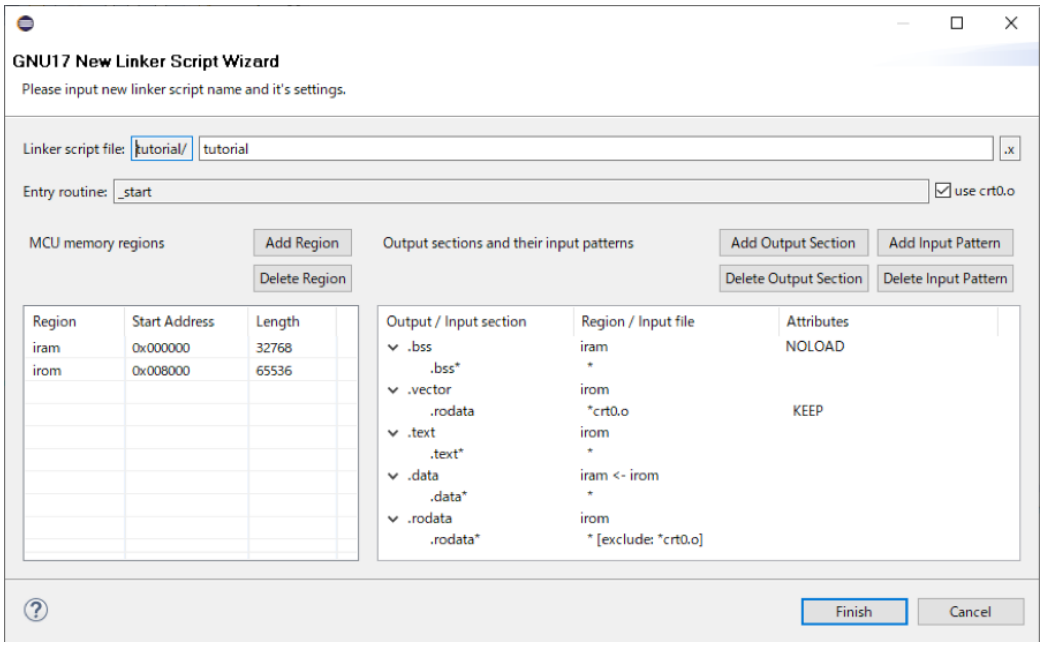
To create a new linker script file

Step 1: Select [GNU17 Linker Script] from the (New) drop-down list* in the toolbar.

* This can also be selected from the [File] menu > [New] and the (New C/C++ Project) drop-down list in the toolbar.



The [GNU17 New Linker Script] wizard will start. At this time, default settings are displayed.



Step 2: Set up the items described below and then click the [Finish] button.

The generated linker script file must be selected in the [GNU17 Setting] page of the project's [Properties] dialog box.

Setup items

(1) Linker script file:

Enter the linker script file name. The linker script file will be generated with the name "(specified file name).x" in the project folder currently selected.

(2) Entry routine:

Enter the program start address to be set with the ENTRY command.
When using the boot processing library "crt0.o," enter "_start" and select the [use crt0.o] check box.

(3) MCU memory regions

Enter the memory configuration (region names, start addresses, capacities) to be described with the MEMORY command. The following is the default definition:

iram : ORIGIN = 0, LENGTH = 32K
irom : ORIGIN = 0x8000, LENGTH = 64K

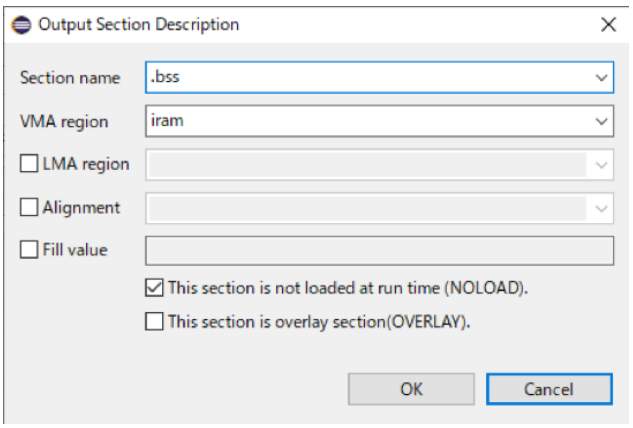
To modify the ORIGIN or LENGTH setting of iram or irom, select the value displayed in the [Start Address] or [Length] column by clicking on it and enter the new value.
To add a new memory region, click the [Add Region] button. A region named "region*X*" is added to the memory region list. Modify the contents according to the region to be added.

(4) Output sections and their input patterns

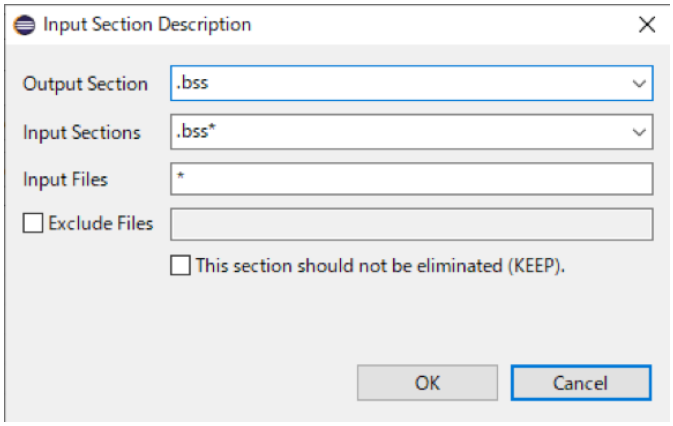
Enter the output and input sections to be described with the SECTIONS command.

To change the output and input sections

- Double-click on the name of the output section to be changed to bring up the [Output Section Description] dialog box. Edit the output section name, VMA region name, LMA region name (optional), alignment (optional), and value to be filled (optional) in this dialog box.

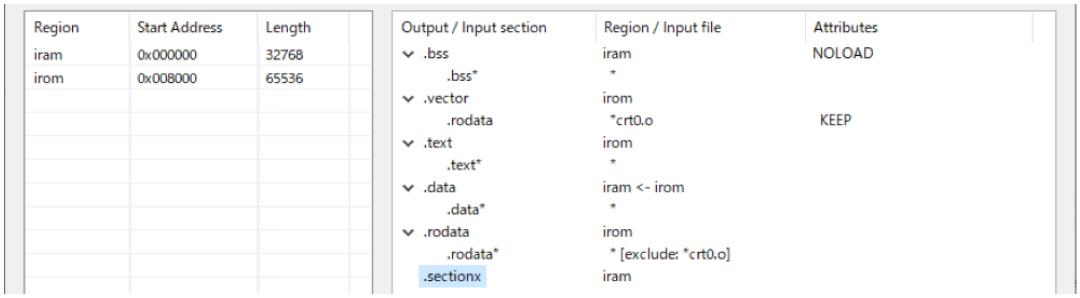


- Double-click on the name of the input section to be changed to bring up the [Input Section Description] dialog box. Edit the input section name and input file name in this dialog box.

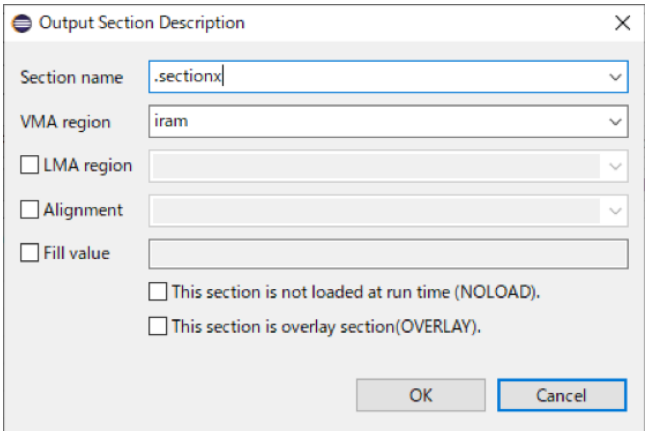


To add output and input sections

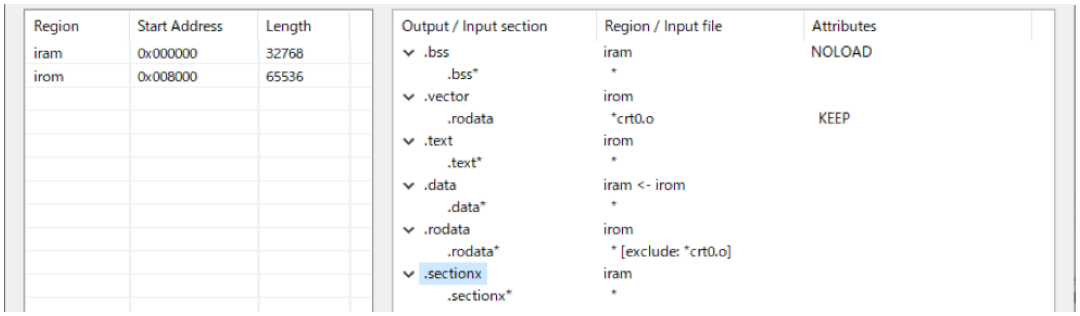
1. Click the [Add Output Section] button. A section definition with the output section name = ".section*X*" and the VMA region name = "iram," is added.



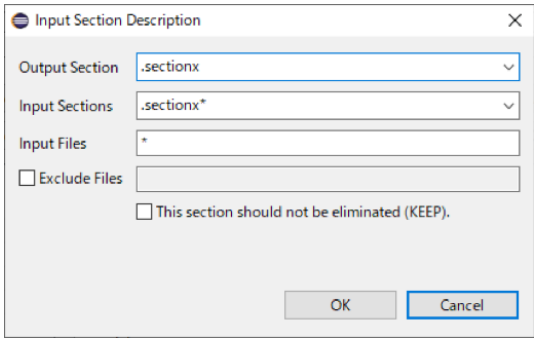
2. Edit the output section name, VMA region name, LMA region name (optional), alignment (optional), and value to be filled (optional) in the [Output Section Description] dialog box brought up by double-clicking on ".section*X*."
   Use the check boxs to define the NOLOAD and OVERLAY attributes for the output section.



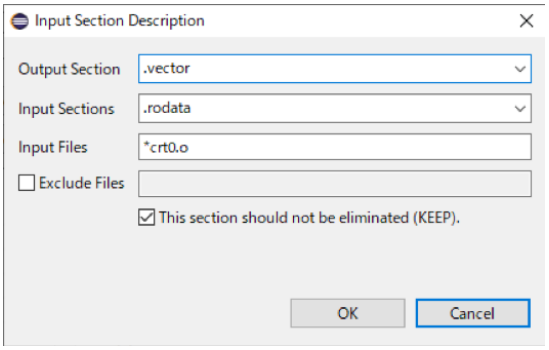
3. After selecting an output section name, clicking the [Add Input Pattern] button adds an input section that corresponds to the output section.



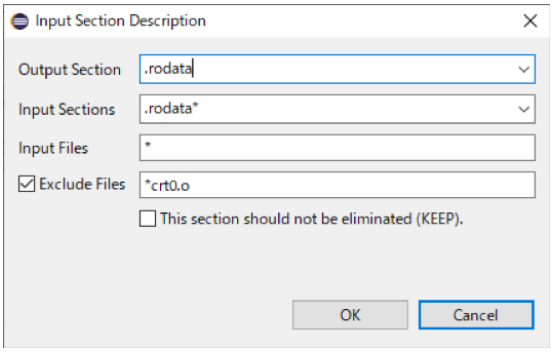
4. Edit the input section name and input file name in the [Input Section Description] dialog box brought up by double-clicking on the input section name.



KEEP and EXCLUDE_FILE should be defined by setting as follows:



"KEEP (*crt0.o(.rodata))" definition



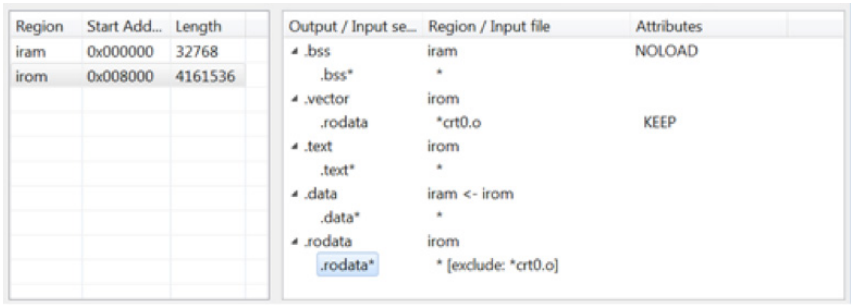
"*(EXCLUDE_FILE (*crt0.o) .rodata)" definition

Repeat the above operations for the number of sections to be defined.
For the contents that cannot be defined in the wizard, edit the generated linker script file with the editor.

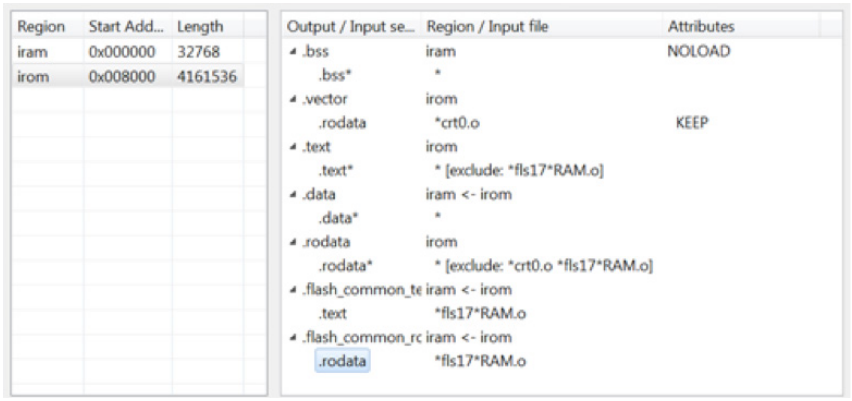
To delete output and input sections

- To delete an output section, click the [Delete Output Section] button after selecting the output section name to be deleted.
- To delete an input section from the output section to which it belongs, click the [Delete Input Pattern] button after selecting the input section name to be deleted.

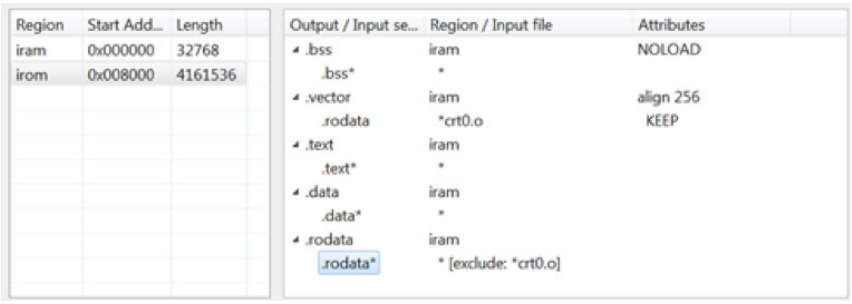
The following shows [GNU17 New Linker Script] wizard setting examples:



Example of definition equivalent to the standard linker script shown in Section A.2, “Linker Script”



Example of linker script definition when using self-programming library



Example of linker script definition when executing program in RAM

Appendix B LCD Panel Simulator

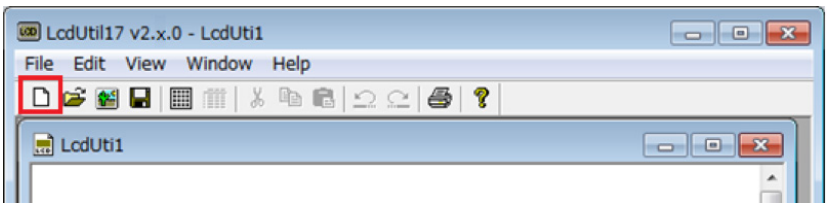
The debugger has an LCD panel simulator function to simulate an LCD panel display, which changes according to the program executed on the actual target board, on the PC. This makes it possible to monitor the LCD panel display on the PC screen even if the target board has no LCD panel mounted.

B.1 How To Configure an LCD Panel Using the LCD Panel Customize Tool (LCDUtil17)

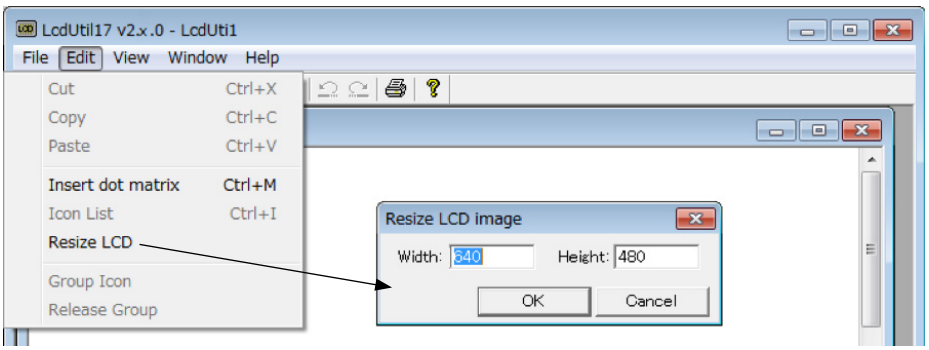
Use the LCD panel customize tool (LCDUtil17) to configure the LCD panel to be simulated on the LCD panel simulator.
To launch LCDUtil17, click the [Launch LCD Utility] button on the [C/C++] perspective.

To design a dot-matrix LCD panel simulation screen

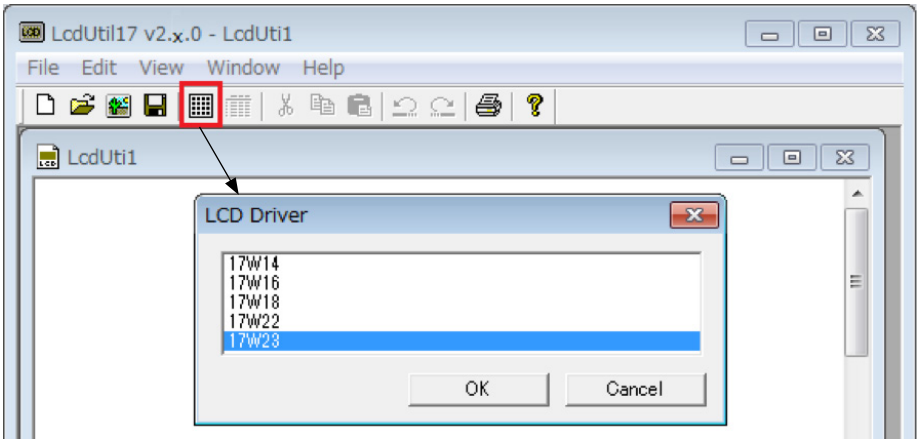
Step 1: Click the (New) button on the [LcdUtil17] window.



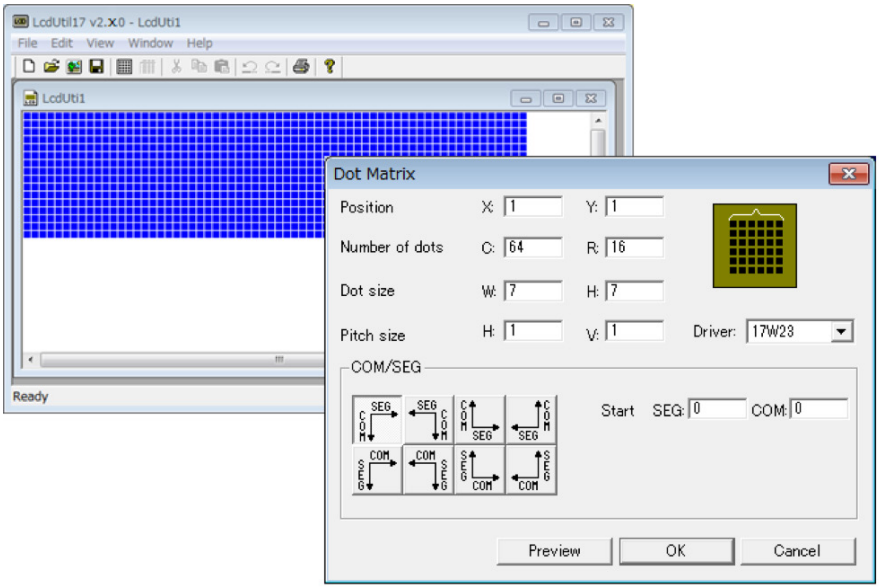
Step 2: Select [Resize LCD] from the [Edit] menu to bring up the [Resize LCD image] dialog box. Enter the LCD panel size in the dialog box, and then click the [OK] button.



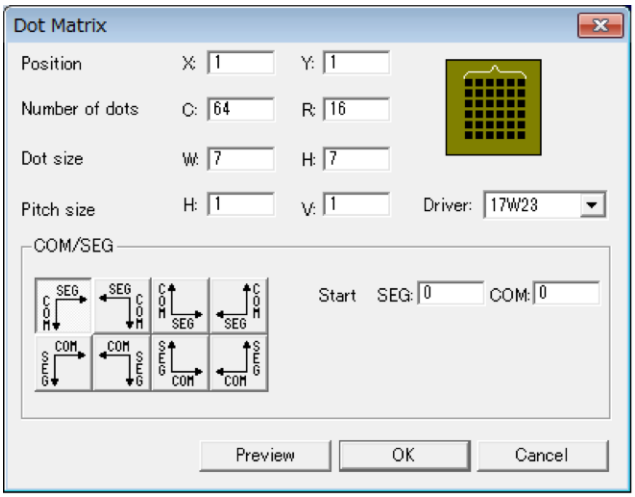
Step 3: Click the (Dot Matrix) button to bring up the [LCD Driver] dialog box. Select a model name in the dialog box and click the [OK] button.



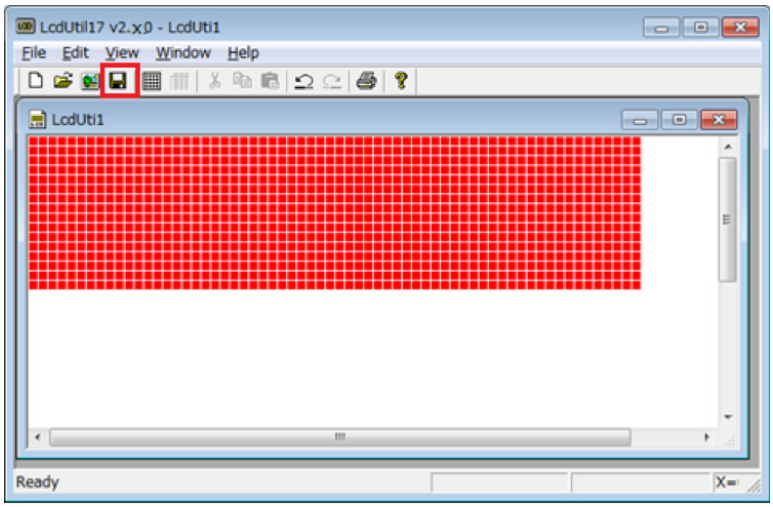
Step 4: Set up the detailed LCD panel design parameters. Select the COM/SEG directions and enter the parameters by using the layout diagram displayed on the right of the [Dot Matrix] dialog box as a reference, and then click the [Preview] button.



Step 5: Click the [OK] button after confirming the previewed layout.



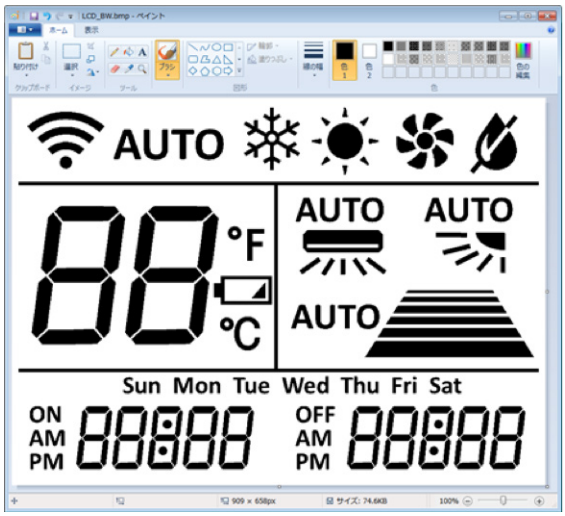
Step 6: Click the (Save) button to save the created LCD panel design data to a file (*.lcd).



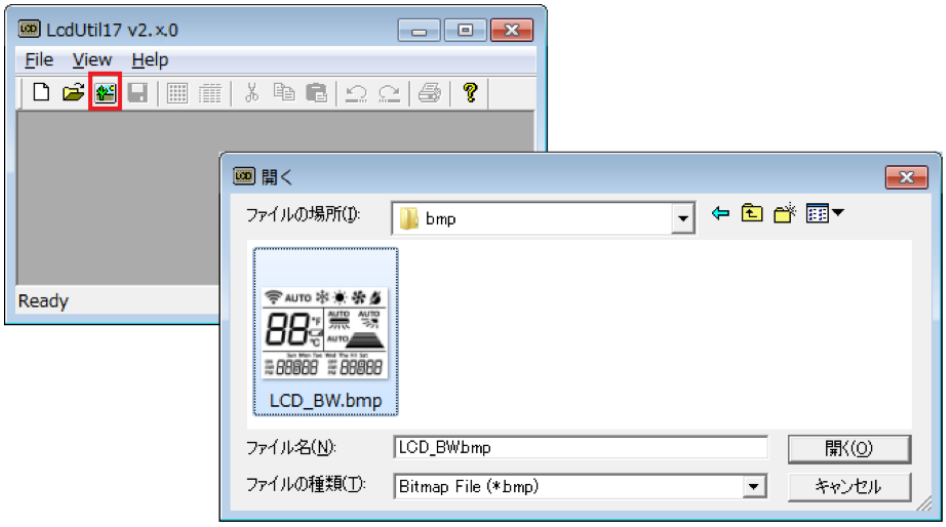
To design a segment LCD panel simulation screen

Step 1: Make a segment layout diagram in black and white using a paint or graphic tool and save it as a monochrome bitmap file (*.bmp).

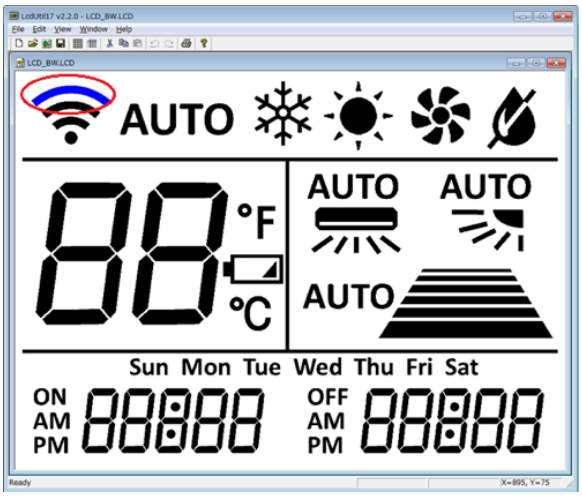
Note: Do not use a color other than black and white.
Keep enough space between the patterns if they will be registered as different segments. Otherwise, they are recognized as one segment.



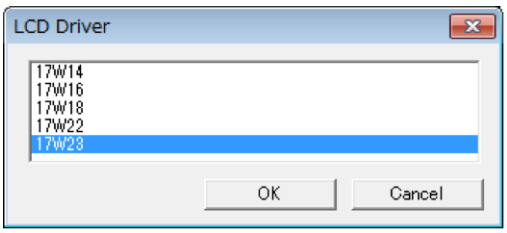
Step 2: Click the (Bitmap) button on the [LcdUtil17] window to load the bitmap file.



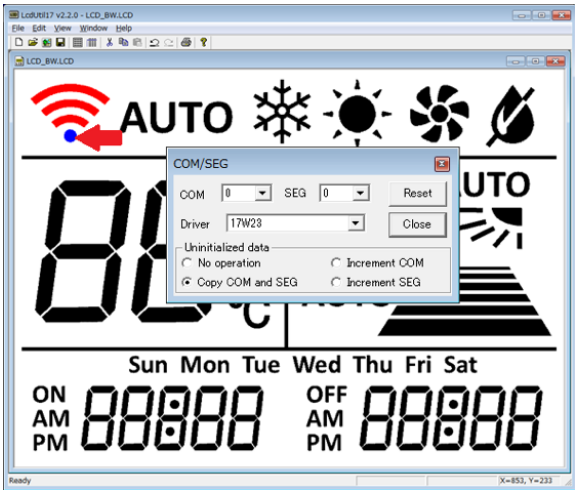
Step 3: Double-click on a segment pattern.



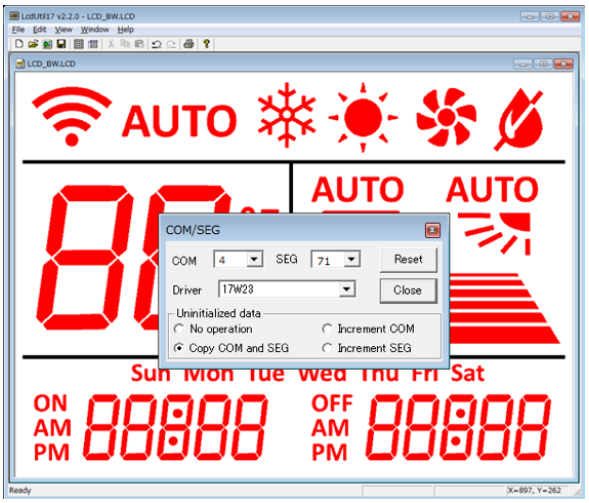
Step 4: Select a model name in the [LCD Driver] dialog box that appears, and then click the [OK] button.



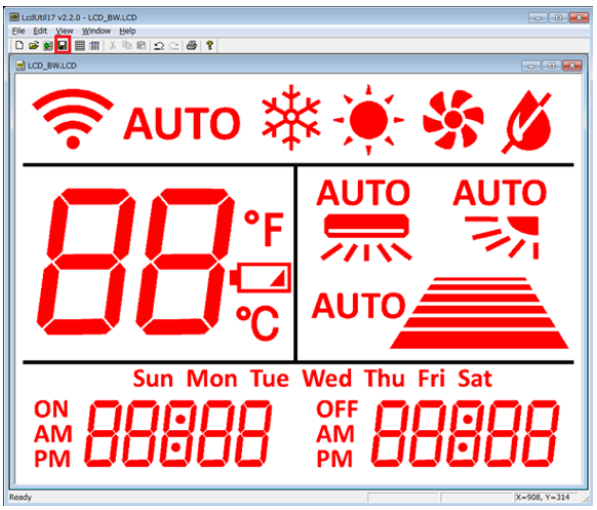
Step 5: Select each segment pattern and specify the COM and SEG numbers to be assigned. (It is possible to specify the same COM and SEG numbers to two or more segment patterns.)



Step 6: Click the (Close) button to close the [COM/SEG] dialog box after completing all the segment settings.



Step 7: Click the (Save) button to save the created LCD panel design data to a file (*.lcd).



B.2 How To Use the LCD Panel Simulator

The LCD panel simulator simulates an LCD panel using an LCD panel file (.lcd) created by the LCD panel customize tool (LCDUtil17). The debug mode must be set to an ICDmini mode, as this function works together with an actual target board.

Advance preparation

Step 1: Create the LCD panel data for simulation using the LCD panel customize tool (LCDUtil17). (Refer to the previous section.)

Step 2: Check to see if the necessary files exist in the device information folder (GNU17V3/mcu_model/17xxx).

- essim17.ini
- essim17_user_def.ini
- lcdDisplaySim17xxx.dll

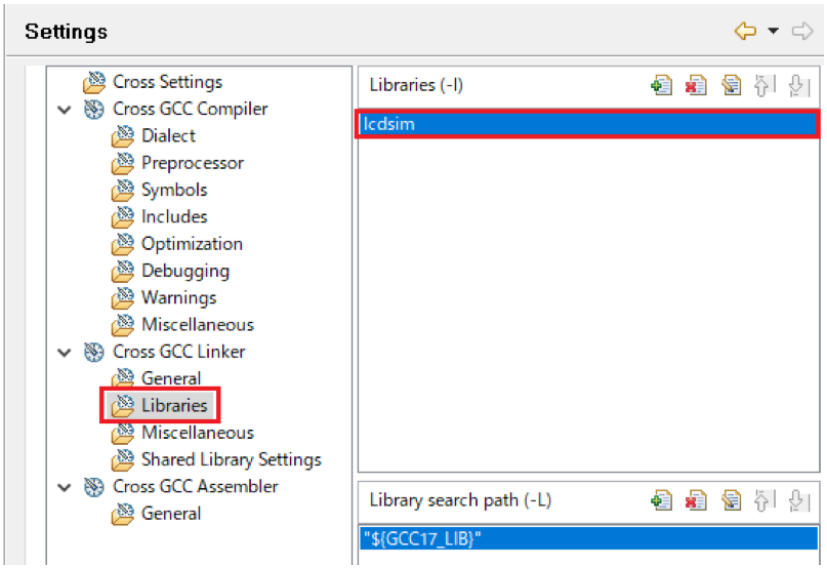
Step 3: Edit the GDB command file.

The debug mode must be set to ICDmini mode, therefore, edit either "gdbmini2.ini" or "gdbmini3.ini" according to your debug environment. The following shows an example using "gdbmini3.ini."

```
# Initial GDB command file for ICDmini3
# set output-radix 16
c17 model_path C:/EPSON/GNU17V3/mcu_model
c17 model 17xxx
target icd icdmini3
load
# Please uncomment following commented out lines to enable STDOUT while debugging.
# c17 stdout 1 WRITE_FLASH WRITE_BUF
# Please uncomment following commented out lines to enable STDIN while debugging.
# c17 stdin 1 READ_FLASH READ_BUF
# Please uncomment following commented out lines to enable LCD panel simulator while debugging.
c17 lcdsim on                                   ... Enable the c17 lcdsim command by uncommenting.
```

Editing the user program

Step 4: Add a library.
Select the target project in the [Project Explorer] view and select [Properties] from the [Project] menu (or context menu that appears by right-clicking) to bring up the [Properties] dialog box. Select [C/C++ Build] > [Settings] from the property list to open the [Tool Settings] tab page. Select [Libraries] from the list under [Cross GCC Linker] and add the LCD panel simulator library "lcdsim."



If the project is created using GNU17 Ver. 3.2 or a later version, this library has already been registered.

Step 5: Edit the target program file and add the include file (lcdsim.h) for the LCD panel simulator library.

```
#include “lcdsim.h”
```

Step 6: Append the LCD panel simulator display update function (lcdupdate();) to the statements that operates an LCD driver control register. This reflects the contents of the altered LCD driver control register to the LCD panel simulator. The following shows an example for S1C17W22 use:

```
Example: LCD24DSP.DSPC = 0x2          // LCD all on
         lcdsimUpdate();              // Update the display in the LCD window.
         LCD24DSP.DSPC = 0x0          // LCD all off
         lcdsimUpdate();              // Update the display in the LCD window.
```

∗ The LCD24DSP.DSPC bit controls the LCD display status.

Note: The final operation check must be performed after mounting the actual LCD panel on the target board. At this time, the appended statements (#include statement for the LCD panel simulator library and LCD panel simulator display update function call statements) should be deleted from the program.

Launching the LCD panel simulator

Step 7: Connect an ICDmini (S5U1C17001H*) and the target system to the PC. The following shows an example using ICDmini Ver. 3:

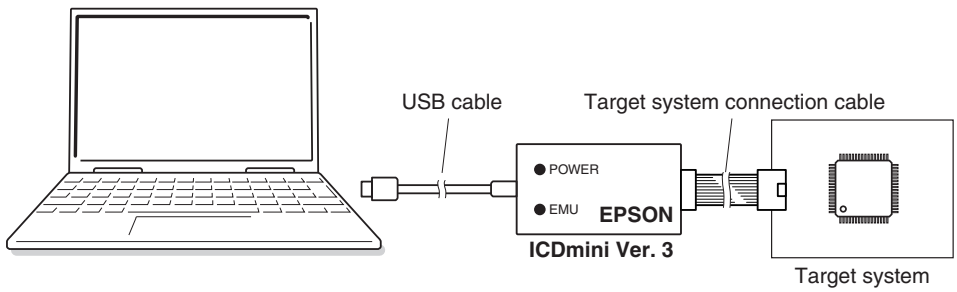
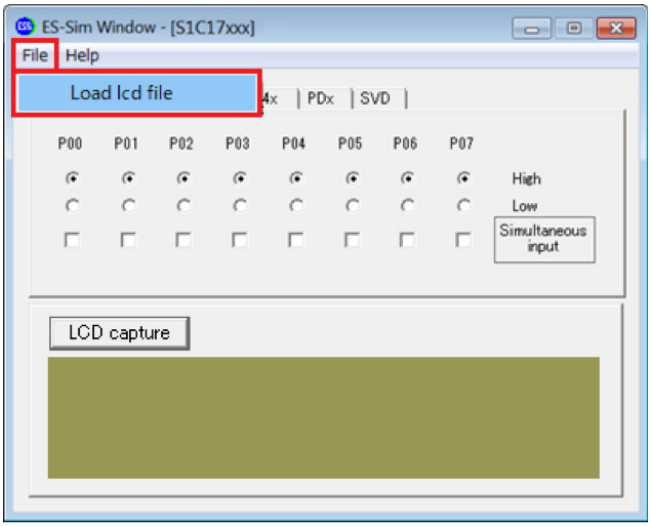


Figure B.2.1 Example of Debugging System Using ICDmini Ver. 3

Step 8: Launch the debugger from the [Debug Configurations] dialog box.

For more information on the debugger, refer to Section 2.7, "Debug."

Step 9: When the debugger starts up, the peripheral circuit simulator (ES-Sim17) window opens. Select [Load lcd file] from the [File] menu on the [ES-Sim] window and open the LCD file (.lcd) that has been created using LCDUtil17.



This package contains a sample LCD file. This file can be used as reference.

C:\EPSON\GNU17V3\utility\LcdUtil17\sample

Step 10: Execute the program.

Click the (Resume) button (or select [Resume] from the [Run] menu). When the LCD panel simulator display update function is called, the [ES-Sim] window displays the simulated LCD panel.

Appendix C Localization (For Reference)

The GNU17 IDE is developed based on the Eclipse IDE for C/C++ Developers Package and the user interface of the workbench installed uses English for its display. If you need to localize it, install a language pack.
The following shows an example how to install a language pack from the Babel project to the GNU17 IDE.

Step 1: Access to the Babel project archive site shown below.

http://archive.eclipse.org/technology/babel/babel_language_packs/R0.15.1/mars/mars.php

Note: This URL may be changed. Please confirm it on the Eclipse website.

Step 2: Download the plugin files for the desired language.

The following shows the example of the files for installing Japanese language pack:

- BabelLanguagePack-eclipse-ja_4.5.0.v20171231064042.zip
- BabelLanguagePack-tools.cdt-ja_4.5.0.v20171231064042.zip

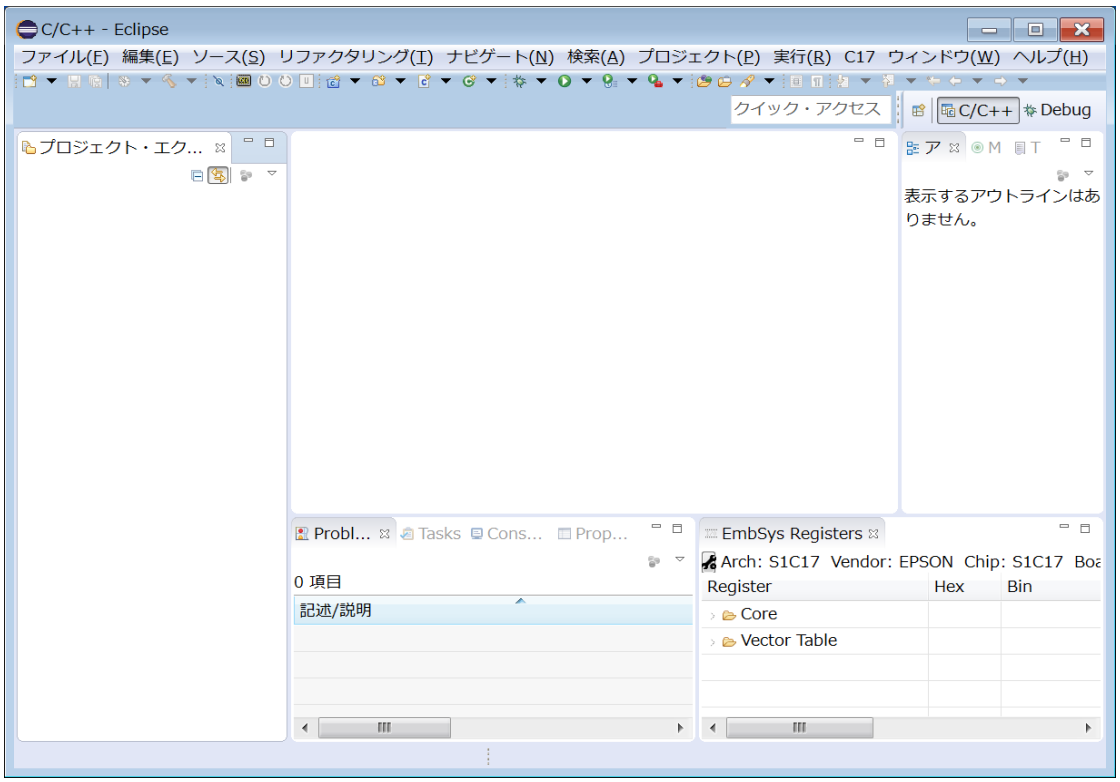
Step 3: Unzip the downloaded files and copy the contents in the extracted eclipse folder to the folder shown below.

C:\EPSON\GNU17V3\eclipse

Step 4: Relaunch the IDE to apply changes. Note that the IDE will take a much longer time at first startup after the language pack is installed.

Note: When the plugin cannot be installed, launch the IDE from the Windows command prompt using the command shown below.
C:\EPSON\GNU17V3\eclipse>eclipse -clean

The menu and other text will be displayed in the installed language after the IDE restarts.
Example: Japanese



To launch the IDE with the language specified

The following shows how to restore the language to English after a language pack is applied or to launch the IDE with a specific language when one or more language packs have been applied.

1. Specifying in the start-up command

To specify a specific language sporadically, launch the IDE from the command prompt in the "eclipse.exe -nl (language)" format.

Example: English

eclipse.exe -nl en

To use different languages, make the shortcuts of "eclipse.exe" for the number of languages and add a -nl option to the [Target:] in the properties of the shortcuts.

Example: C:\EPSON\GNU17V3\eclipse\eclipse.exe -nl en

2. Specifying in eclipse.ini

The language usually used should be specified in the "eclipse.ini" file that exists in the "C:\EPSON\GNU17V3\eclipse" directory as in the format below.

-Duser.language=(language)

-Duser.country=(country)

Example: English

-Duser.language=en

-Duser.country=US

Revision History

Attachment-1

Rev. No.	Date	Page	Category	Contents
Rev. 1.0	2019/09/09	All	New	New establishment
Rev. 1.1	2023/04/11	P56 to P62	Revision	Modified the explanation of Section 3.2, “Importing a GNU17 Ver. 2.x Project.”
		P67 to P68	Revision	Added Section A.3, “Linker Script Example.”

EPSON

International Sales Operations

America

Epson America, Inc.

Headquarter:
3131 Katella Ave.,
Los Alamitos, CA 90720, USA
Phone: +1-800-463-7766

San Jose Office:
2860 Zanker Road Suite 204,
San Jose, CA 95134, USA
Phone: +1-800-463-7766

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.

4th Floor, Tower 1 of China Central Place, 81 Jianguo Road,
Chaoyang District, Beijing 100025, China
Phone: +86-10-8522-1199 Fax: +86-10-8522-1120

Shanghai Branch

Room 601-603, Building A One East, No. 325 East Longhua
Road, Shanghai 200023, China
Phone: +86-21-5330-4888 Fax: +86-21-5423-4677

Shenzhen Branch

Room 804-805, 8 Floor, Tower 2, Ali Center, No. 3331
Keyuan South RD (Shenzhen bay), Nanshan District,
Shenzhen 518054, China
Phone: +86-755-3299-0588 Fax: +86-755-3299-0560

Epson Taiwan Technology & Trading Ltd.

15F, No. 100, Songren Rd, Sinyi Dist, Taipei City 110, Taiwan
Phone: +886-2-8786-6688

Epson Singapore Pte., Ltd.

438B Alexandra Road,
Block B Alexandra TechnoPark, #04-01/04, Singapore 119968
Phone: +65-6586-5500 Fax: +65-6271-7066

Epson Korea Co., Ltd

10F Posco Tower Yeoksam, Teheranro 134 Gangnam-gu,
Seoul, 06235, Korea
Phone: +82-2-3420-6695

Seiko Epson Corp.
Sales & Marketing Division

MD Sales & Marketing Department

JR Shinjuku Miraina Tower, 4-1-6 Shinjuku, Shinjuku-ku,
Tokyo 160-8801, Japan

Document Code: 413880601
First Issue September 2019
Revised May 2023 in JAPAN Ⓛ