User's Manual

RZ/G Series

RZ/G Linux Platform Development Environment User's Manual

for the Linux Customization Tool, the Verification and Analysis Tools

Target Devices

RZ Family

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.2.11 Oct 2018

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

How to Use This Manual

Purpose and Target Readers

This manual provides users of the development environment services for use with the RZ/G Linux platform a description of how to use the RZ/G Linux platform development environment. Applying the information in this manual requires knowledge of the fundamentals of software development, including operating systems and programming.

Particular attention should be paid to the precautionary notes when using RZ/G-series products. These notes occur within the body of the text, at the end of each section, and in the Usage Notes section.

The following documents are provided for RZ/G-series products.

Document Type	Outline	Document Title
Development environment	Gives an outline of how to use the development environment.	Development Environment Guide
Development tools	Describes how to use RZ/G Linux platform Development Environment, consisting of the Linux Customization tool, Verification tool, and Analysis tool.	RZ/G Linux Platform Development Environment User's Manual for the Linux Customization Tool, the Verification and Analysis Tools
	Describe how to use Linux Customization tool and Verification tool, based on examples.	RZ/G Linux Platform Development Environment Tutorial Manual
	Describes how to set up the local build server for the RZ/G Linux Platform Development Environment.	Build Server Setup Manual for RZ/G Linux Platform Development Environment
	Describes how to create test programs.	Guide for Creating Test Programs for the Software Verification Tool
Hardware user's manual	Describes common specifications.	RZ/G Series User's Manual: Hardware
	Describes specifications specific to the RZ/G1H.	RZ/G1H User's Manual: Hardware
	Describes specifications specific to the RZ/G1M	RZ/G1M User's Manual: Hardware
	Describes specifications specific to the RZ/G1N	RZ/G1N User's Manual: Hardware
	Describes specifications specific to the RZ/G1E	RZ/G1E User's Manual: Hardware
	Describes specifications specific to the RZ/G1C	RZ/G1C User's Manual: Hardware

Abbreviation

Abbreviation	Spelling Out	Description
BSP	Board Support Package	In general, “BSP” means the "Board Support Package" (a software component that allows you to run the operating system on a specific hardware platform). But in this document and each tool described in this document, the "Verified Linux Package" provided by our company is referred to as "BSP" depended on each context.
OpenGL ES	OpenGL for Embedded Systems	This is a subset of OpenGL for embedded systems. It serves as the programming interface for graphics processing.
Qt	Qt	Cross-platform application framework in the form of a widely known GUI toolkit
CMMI	Capability Maturity Model Integration	Framework for use in evaluating or improving software development processes in organizations and companies. It is defined in terms of the degrees of maturity of development processes.
e^2 studio	e^2 studio	Eclipse-based integrated development environment which supports devices from Renesas

All trademarks and registered trademarks are the property of their respective owners.

- OpenGL is a registered trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.
- Qt is a trademark or registered trademark of The Qt Company Ltd. and its subsidiaries in the United States and other countries.
- CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.
- Eclipse is a registered trademark of the Eclipse Foundation, Inc. in the United States, other countries, or both.
- Git and the Git logo are either registered trademarks or trademarks of Software Freedom Conservancy, Inc., corporate home of the Git Project, in the United States and/or other countries.
- The official name of Windows is Microsoft® Windows® Operating System.
- Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
- Linux is a registered trademark or trademark of Linus Torvalds in the United States and other countries.
- The symbols for trademark and registered trademark (® and ™) may be omitted in this manual.

Contents

1. Overview 1

1.1 Development Environment Services 1

1.2 Operating Environment 3

2. Setting up RZ/G Linux Platform Tools 4

2.1 Installation Method 4

2.1.1 Installing the e^2 studio 4

2.1.2 Installing the MSYS2 8

2.1.3 Installing the Visual C++ Run-time Components 12

2.1.4 Installing the Serial Driver 12

2.1.5 Installing the Plug-ins 13

2.1.6 Creating a Folder for Storing Tool Data 17

2.2 Uninstallation Method 19

2.2.1 Uninstalling the Plug-ins 19

2.3 Confirming the Installed Plug-in Version 20

3. Handling the Linux Customization Tool 21

3.1 List of Functions 21

3.2 Application Examples 23

3.2.1 SD Card Boot on the Target Board 23

3.2.2 eMMC Boot on the Target Board 24

3.2.3 Combining User Program Build with SD Card Boot 25

3.2.4 Directly Writing a User Program to the Target Board 26

3.2.5 Customizing a Yocto Recipe and Registering the Results of Customization with the Git Repository 27

3.2.6 Adding a Yocto Recipe and Registering the Added Yocto Recipe with the Git Repository 28

3.2.7 Customizing the Kernel and Registering the Results of Customization with the Git Repository 29

3.2.8 Customizing U-Boot and Registering the Results of Customization with the Git Repository 30

3.3 Starting Up the e^2 studio 31

3.4 Handling Build Server 33

3.4.1 User Authentication 33

3.5 Creating a Project 34

3.5.1 Creating an Embedded Linux Project 34

3.5.2 Creating an Embedded Linux Custom BSP Project 38

3.5.3 Deleting a Project 47

3.6 Setting Properties 48
3.6.1 Setting Serial Ports 49
3.6.2 Making the Network Settings 50
3.7 Adding 51
3.7.1 Adding User Programs 51
3.7.2 System of Building User Programs 54
3.7.3 Creating Makefile 56
3.8 Building 74
3.8.1 Setting up Build 74
3.8.2 Executing Build 80
3.9 Downloading 84
3.9.1 Downloading Results of Building 84
3.9.2 Downloading the U-Boot File 87
3.10 Writing 89
3.10.1 Writing to an SD Card 89
3.10.2 Writing to an eMMC 92
3.10.3 Writing a User Program 97
3.10.4 Switching Boot Mode 103
3.10.5 Writing the Network Information 106
3.11 Customizing Yocto Recipes 109
3.11.1 Editing Yocto Recipes 109
3.11.2 Adding Yocto Recipe Patches 111
3.11.3 Adding Yocto Recipes 113
3.12 Customizing the Kernel 116
3.12.1 Changing the Kernel Configuration 116
3.12.2 Adding and deleting of the Kernel Configuration function 119
3.12.3 Initializing the Kernel Configuration 124
3.12.4 Editing the Kernel Code 125
3.12.5 Removing the Contents of Editing the Kernel Code 132
3.12.6 Debugging the Kernel 134
3.13 Customizing U-Boot 138
3.13.1 Editing the U-Boot Code 138
3.13.2 Removing the Contents of Editing the U-Boot Code 144
3.13.3 Debugging the U-boot 146
3.14 Git 147
3.14.1 Using Git in the Version-control of Files 147
3.14.2 Registering with the Git 154
3.15 Creating a Custom BSP 158

3.15.1 Creating a new BSP 158
3.16 Creating and Changing an SDK 161
3.16.1 Creating a new SDK 161
3.16.2 Changing an SDK 164
3.17 Editing the local.conf 166

4. Handling the Verification Tool 168

4.1 Starting up the Verification Tool 168
4.2 Tool Settings 169
4.2.1 Settings for Connection with the Target Board 169
4.2.2 Using the Embedded Linux Project to Import the Settings 172
4.2.3 Specifying the Destinations for Storage of the Test Programs and Results of Tests 173
4.2.4 Selections for the Test Environments 175
4.3 Executing Tests 176
4.3.1 Selecting Items to Be Tested 176
4.3.2 Connecting the Target Board 177
4.3.3 Starting the Tests 178
4.3.4 End of Testing and Terminating the Tests 179
4.3.5 Other Operations from the [Test item list] Section 180
4.4 Confirming the Results of Tests 181
4.4.1 Displaying a List of Test Result Histories 181
4.4.2 Deleting Test Results 183
4.4.3 Exporting a Report File 185
4.4.4 Exporting the File for Analysis Tool 186
4.4.5 Selecting Test Items from the Results of Tests 187
4.5 Saving and Restoring the State of Test Item Selection 189
4.5.1 Saving the State of Test Item Selection 189
4.5.2 Restoring the State of Test Item Selection 192
4.5.3 Deleting a State of Test Item Selection 193
4.6 Registering and Editing Test Programs Created by Users 195
4.6.1 Registering Test Programs in a Single Round 195
4.6.2 Editing Configuration of Test Programs 198
4.7 Importing or Deleting the Test Programs for the Software Add-on 203
4.7.1 Importing the Test Program for Software Add-on 203
4.7.2 Deleting the Test Program for Software Add-on 204
4.8 Registering a Set of Test Programs as an Embedded Linux Project 205
4.9 Saving a Set of Test Programs 206
4.10 Editing a parameter for Test Programs 207

5. Handling the Analysis Tool 208
5.1 Settings to the web server 208
5.1.1 Outline of Functions 208
5.1.2 Logging in 209
5.1.3 Description of Functions 210
5.2 Handling the Analysis Tool 216

6. Handling the Traffic Monitor 217
6.1 Starting up the Traffic Monitor 217
6.2 Tool Settings 218
6.2.1 Registering the Linux command for traffic monitoring 218
6.2.2 Settings for Connection with the Target Board 219
6.2.3 Basic Settings 220
6.3 Executing Measurement 221
6.3.1 Starting Measurement 221
6.3.2 Stopping Measurement 222
6.4 Saving a Log of Measured Results in a File 223
6.4.1 Starting the Log 223
6.4.2 Stopping the Log 223
6.4.3 Setting the Log File 224
6.4.4 Measured Data Saved in the Log File 225

RZ/G Linux Platform Development Environment User's Manual for the Linux Customization Tool, the Verification and Analysis Tools

Renesas Microprocessor

R01US0342EJ0211
Rev.2.11
Oct 22, 2018

1. Overview

This manual describes how to control the Linux Customization tool and the Verification and Analysis tools included in the Development environment service provided on RZ/G Linux platform.

The Linux Customization tool and the Verification tool run in the e2studio environment and the Analysis tool runs on a Web browser and can analyze the verification result.

This manual applies to the following version of the tools:

- RZ/G Linux platform Development Environment: Linux Customization Tool Ver. 2.1.x
- RZ/G Linux platform Development Environment: Verification and Analysis Tool Ver. 2.1.x

This manual describes the following items:

- Installing each tool for RZ/G Linux platform Development Environment
- Handling the Linux Customization tool, Verification tool, analysis tool, and traffic monitor

1.1 Development Environment Services

Figure 1-1 is a schematic diagram of the Development environment services, and Table 1-1 describes the components.

Among the components listed in Table 1-1, this manual describes how to set up and use the build system, which is used for user management and server management, and the RZ/G Linux platform development environment (Linux Customization Tool, Verification Tool, and Analysis Tool).

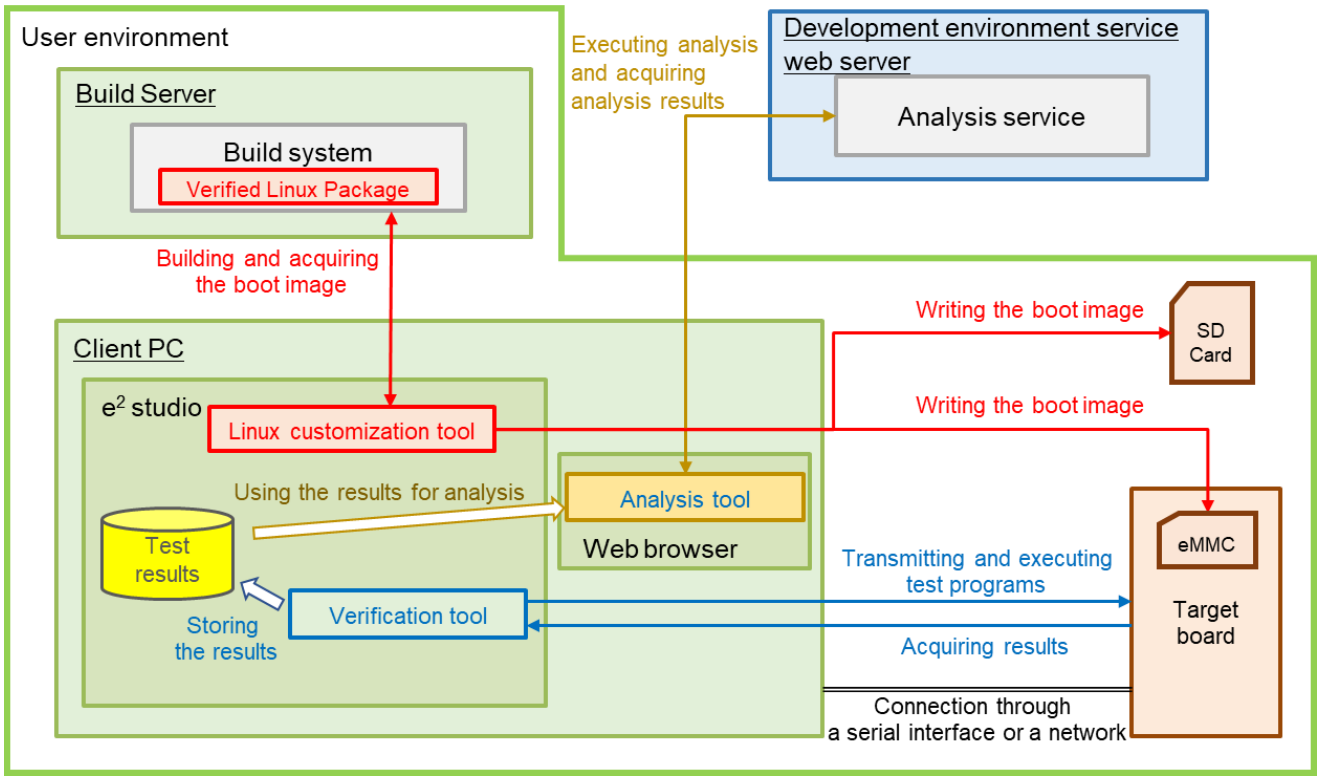


Figure 1-1 Schematic View of Development Environment Services

As shown in Figure 1-1, a Linux environment for the target board will be built using the "Verified Linux Package" (VLP) in this development environment. But because the VLP is referred as "BSP" in this document and the tools described in this document, please note that the BSP may be different with the general term "Board support package".

Table 1-1 Components of Development Environment Services

Component	Description
Build system	Provides build services using "Verified Linux Package", based on customizations modified through the user interface of the "Linux Customization Tool". After the build process, users can download the result file (such as an image file of the SD card) to a client PC. The build system is provided by the Build Server on the server PC prepared by customers.
Analysis service	Provides analysis service for the results of the "Verification tool", through the user interface of "Analysis tool". As analysis results, it will provide some hints for the cause and the solution of the problem, by using the information of failed test items on executing by the "Verification tool". * In the information as the results of "Verification tool", it is included "error or not" and "type of error generated", but not included the customer asset itself (source code, binary, etc.).
e^2 studio	The e^2 studio is an integrated development environment provided by Renesas Electronics (hereinafter referred to as we/us) for the Windows environment. The Linux Customization tool, verification tool, and analysis tool serve as plug-ins for the e^2 studio.
Target board	This refers to the target board to be developed by the user. The Linux Customization tool, verification tool and analysis tool control the Linux bootup, verification, and other required functions for the target board through serial communications or an Ethernet link.
Linux Customization tool	This tool implements the following functions in the e^2 studio environment. Used with the build system, it facilitates quick build and configuration for Linux. • Customizing the BSP based on building the Linux environment and Linux application programs. • Adding middleware and application programs
Verification tool	This tool implements the following functions in the e^2 studio environment. • Testing of the items provided by Renesas (the Linux kernel, Linux BSP, and Qt) which run on the target board; and the execution of user-created test programs • Monitoring traffic to check the state of the target board
Analysis tool	This tool implements the following function through operation on the Web browser. • Displaying hints to assist in solving problems according to the results of using the verification tool to test the items provided by Renesas,

1.2 Operating Environment

The tools for the RZ/G Linux platform development environment are plug-ins for the e^2 studio and run in the following environment.

- Windows 7 (32- or 64-bit version), Windows 8.1 (32- or 64-bit version), Windows 10 (32- or 64-bit version)
- e^2 studio 6.1.0, e^2 studio 7.0.0

2. Setting up RZ/G Linux Platform Tools

2.1 Installation Method

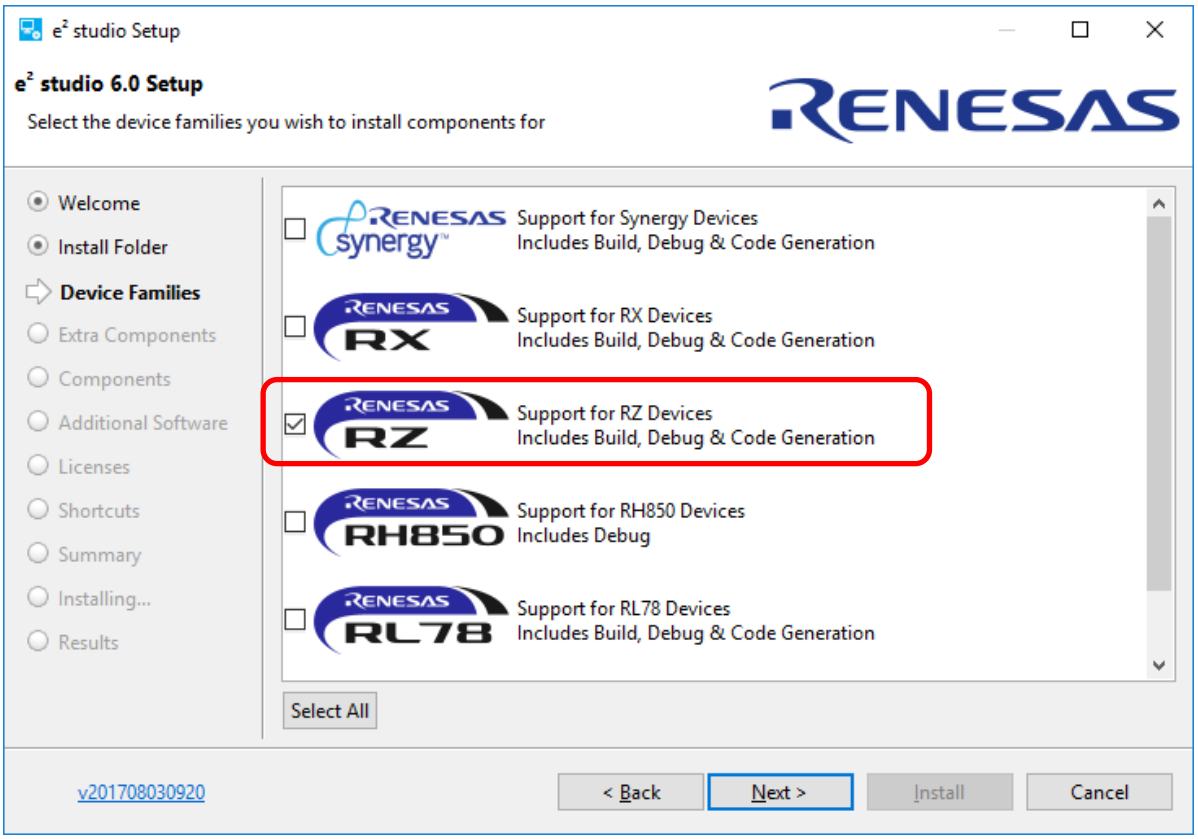
2.1.1 Installing the e² studio

This subsection describes how to install the e² studio (for the versions supported, refer to section 1.2).

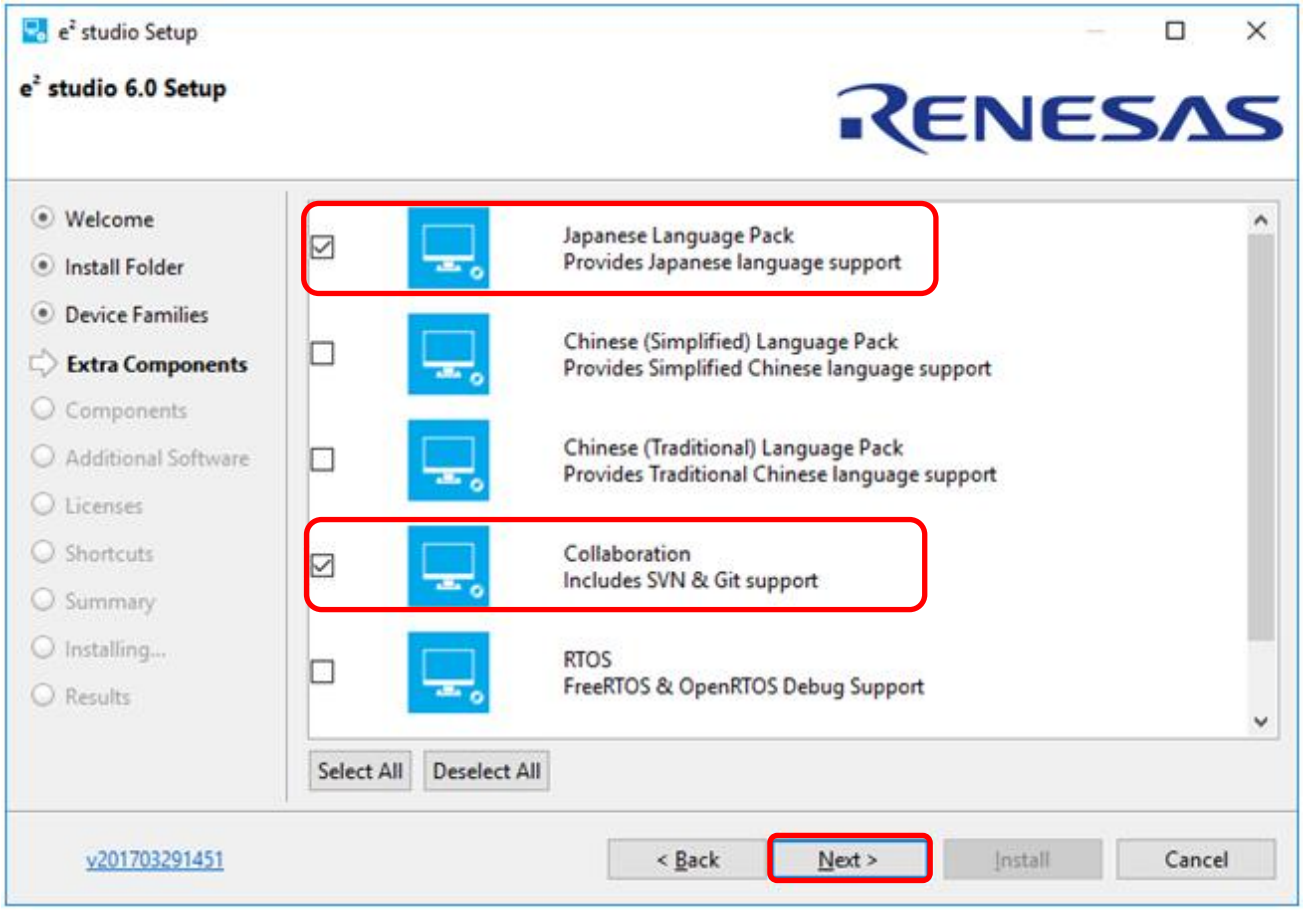
1. Access to the official Renesas Electronics Corp. site and search "e2studio6.1.0" or "e2studio7.0.0". Then download the installer. In following, described by using e² studio 6.1.0 as a sample.
   https://www.renesas.com/en-us/
   If it is not displayed e2 studio6.1.0 in the result of search, Click the [More Results from Tool Downloads] and the link to download page is displayed.
   Note: This tool version is only supported on e² studio 6.1.0 and e2 studio 7.0.0.

2. Double-click on the icon of the downloaded installer file to start the installer.

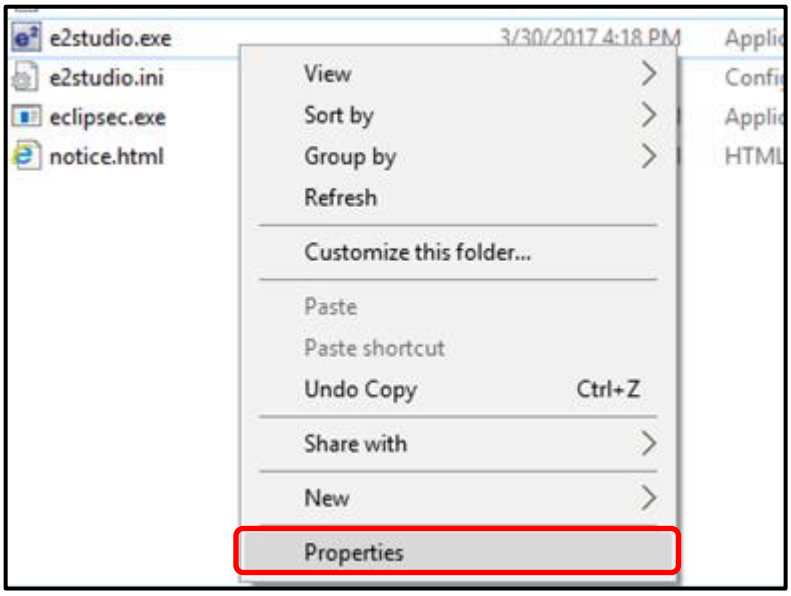
[Note 1 on Installation]
Select [Support for RZ Devices] in the [Device Families] pane.
Note that [Support for RZ/G Devices (Includes Build & Debug)] in the [Device Families] pane is the option for use in cross-compilation on the local PC or remote debugging with GDB. Selecting this does not affect the operation of the tool.



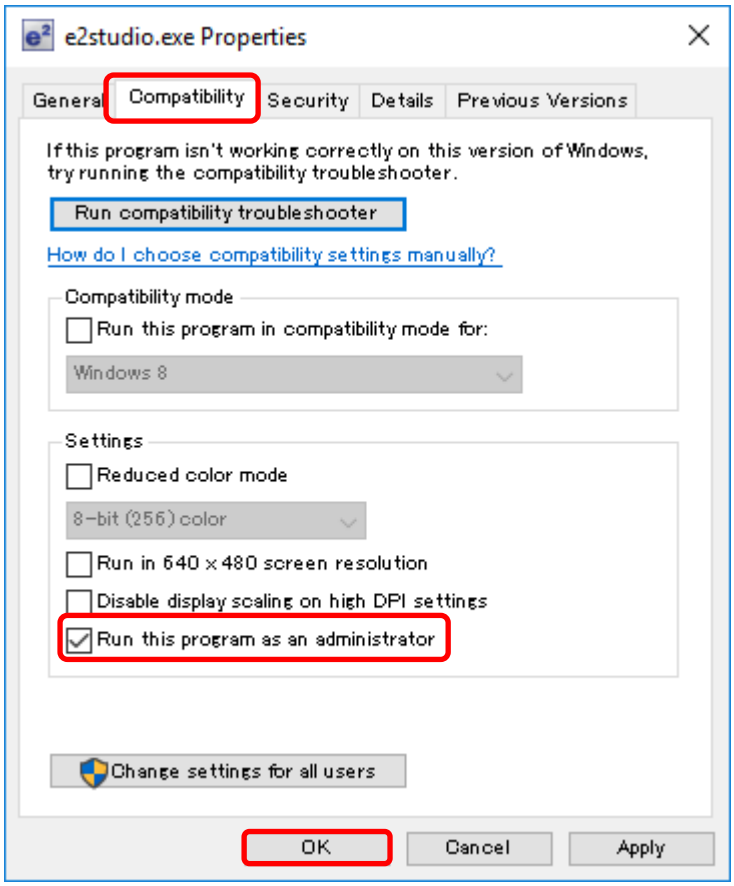
[Note 2 on Installation]
Select [Collaboration (Includes SVN & Git support)] in the [Extra Components] pane.
When using the Japanese tool, select [Japanese Language Pack].



3. Once the installation is completed, make the setting for allowing only users having the Windows administrator privilege to handle the e^2 studio.
   (Note: This is because the administrator privilege is required for writing the built image files to an SD card.)
   Right-click on the [C:\Renesas\e2_studio\eclipse\e2studio.exe] file in Windows File Explorer and select [Properties] from the pop-up menu.



4. When the [e2studio.exe Properties] dialog box appears, select the [Compatibility] tabbed page, click on the checkbox for [Run this program as an administrator] in the [Settings] section, and then click on the [OK] button.



2.1.2 Installing the MSYS2

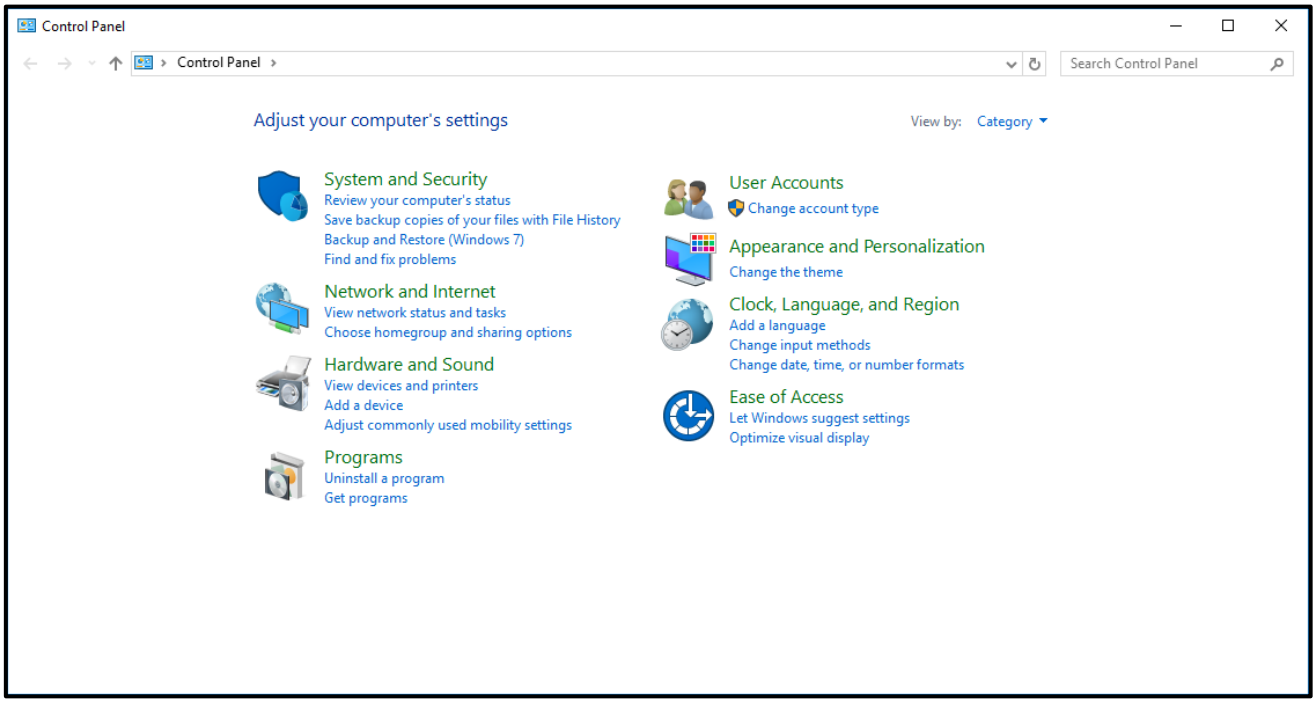
This subsection describes how to install the MSYS2, which is used in writing disk images to an SD card.

1. Download the installer from the website below.
   http://www.msys2.org/

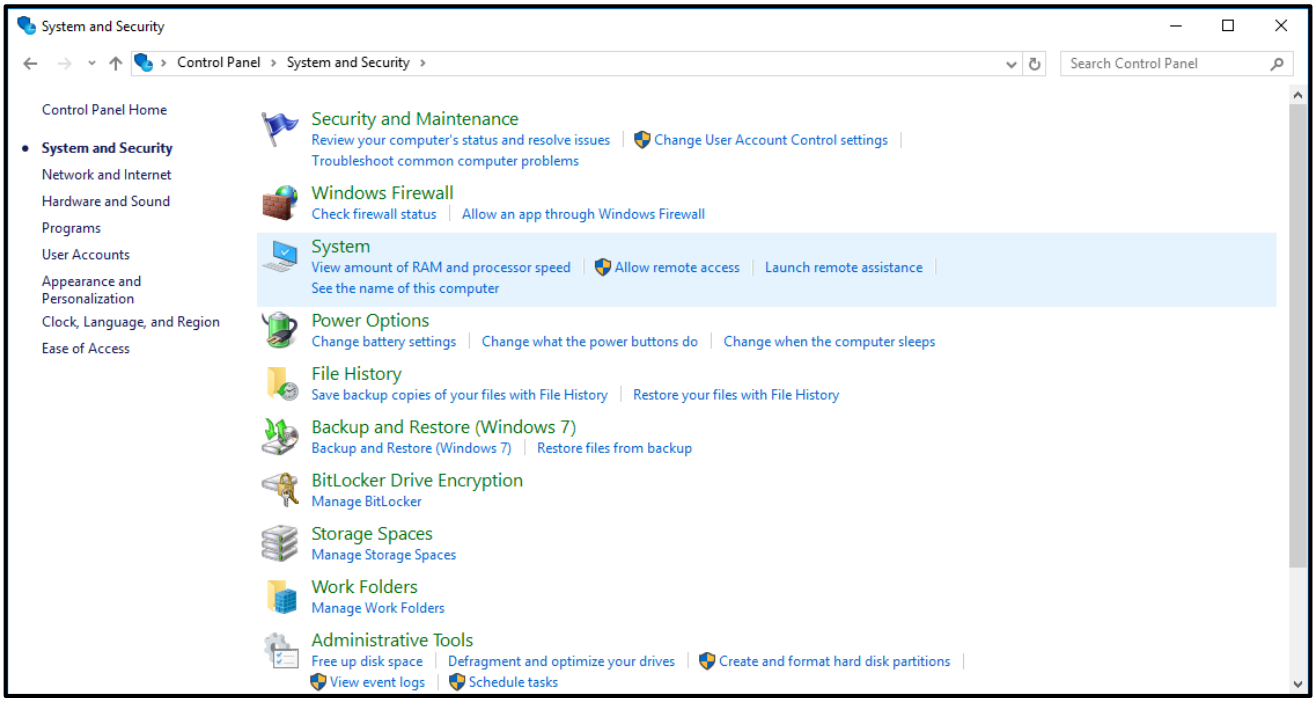
2. Double-click on the icon of the downloaded installer file to start the installer, and install the files according to the setup procedure.

3. Setting environment variables
   To execute MSYS2 commands from the e^2 studio, set environment variables according to the procedure described below.

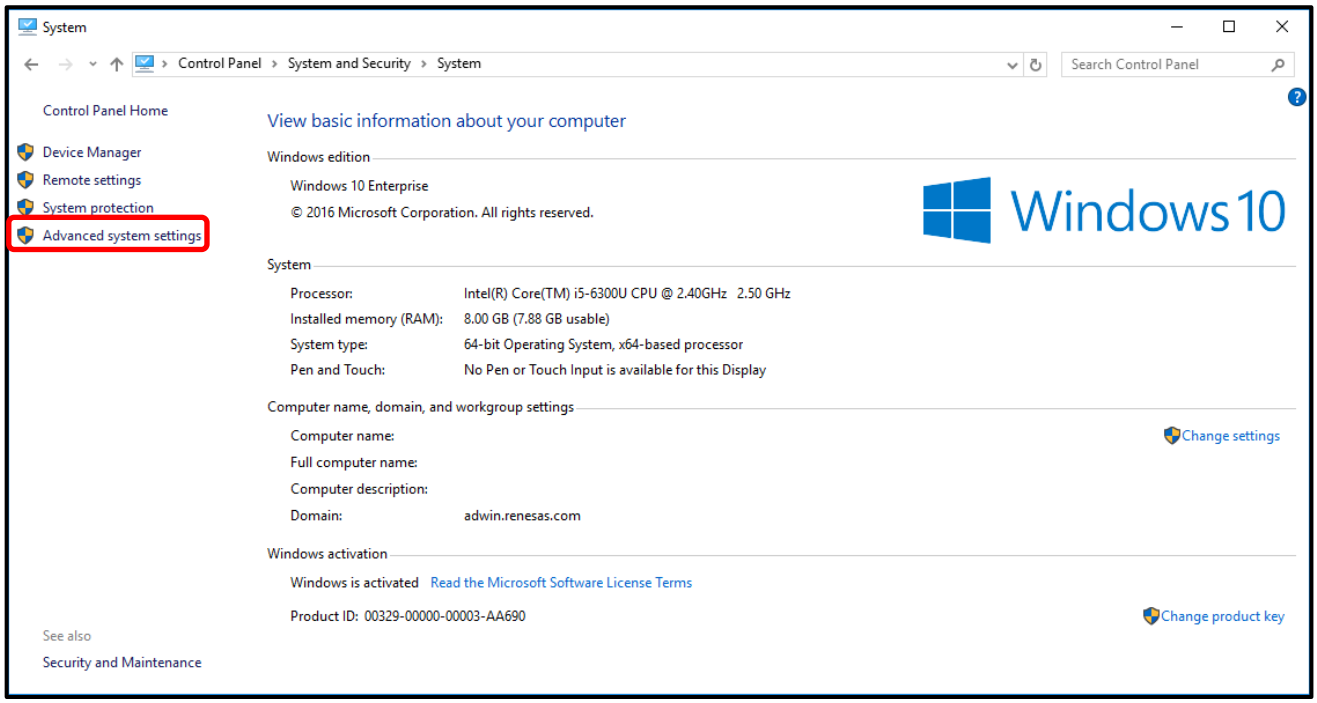
(1) Open [Control Panel], and then click on [System and Security].



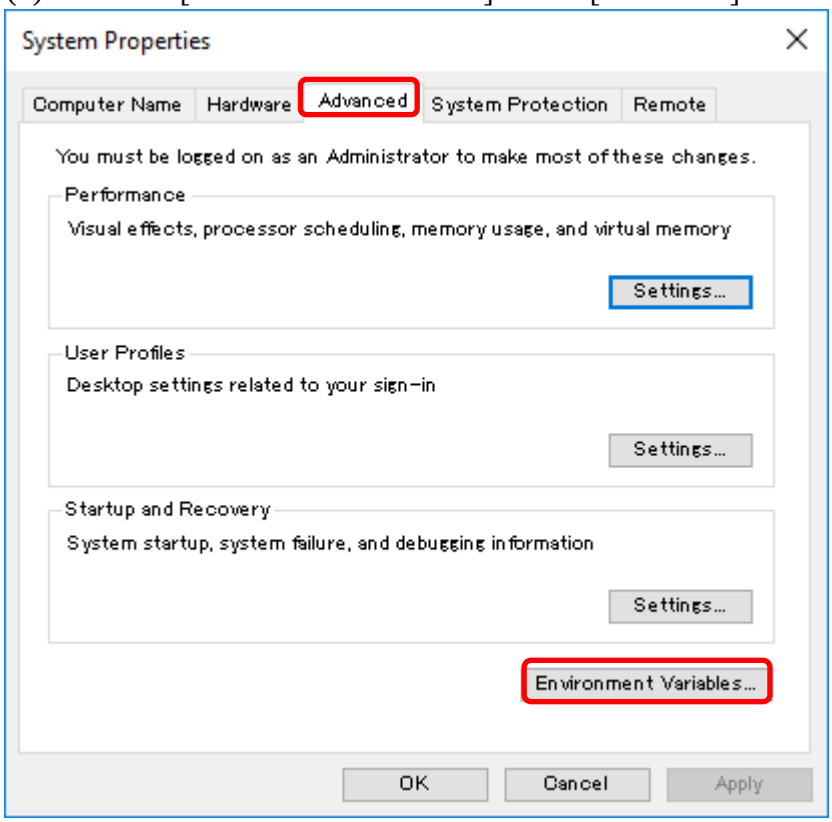
(2) Click on [System].



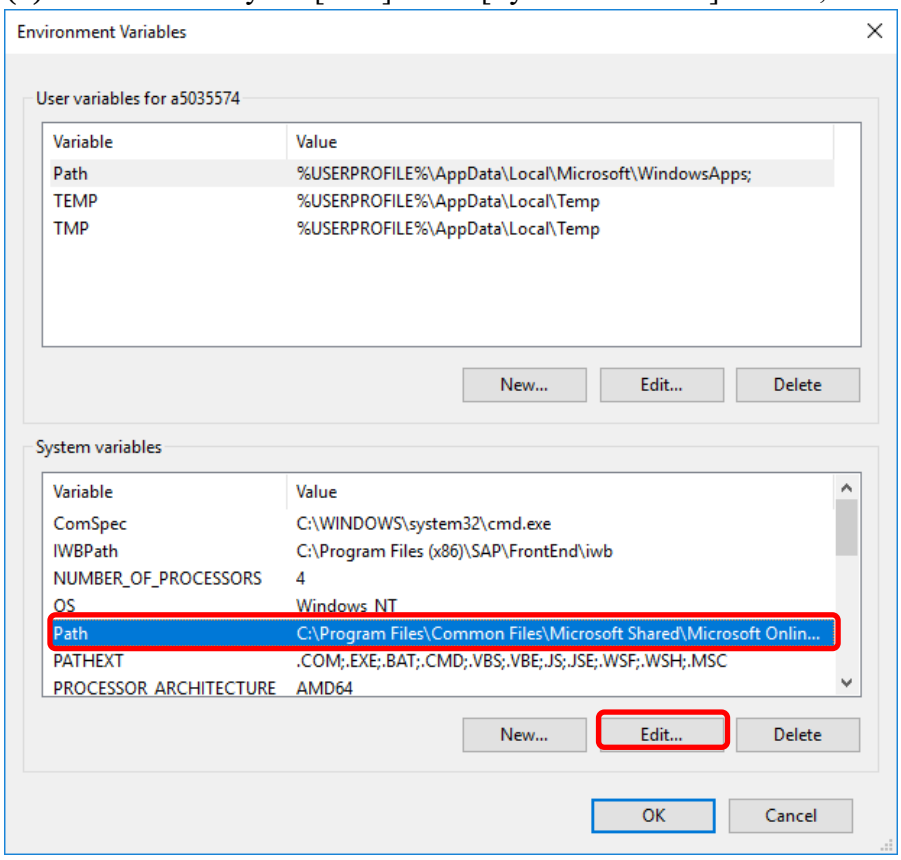
(3) Click on [Advanced system settings] in the left pane.



(4) Click on [Environment Variables] on the [Advanced] tabbed page.

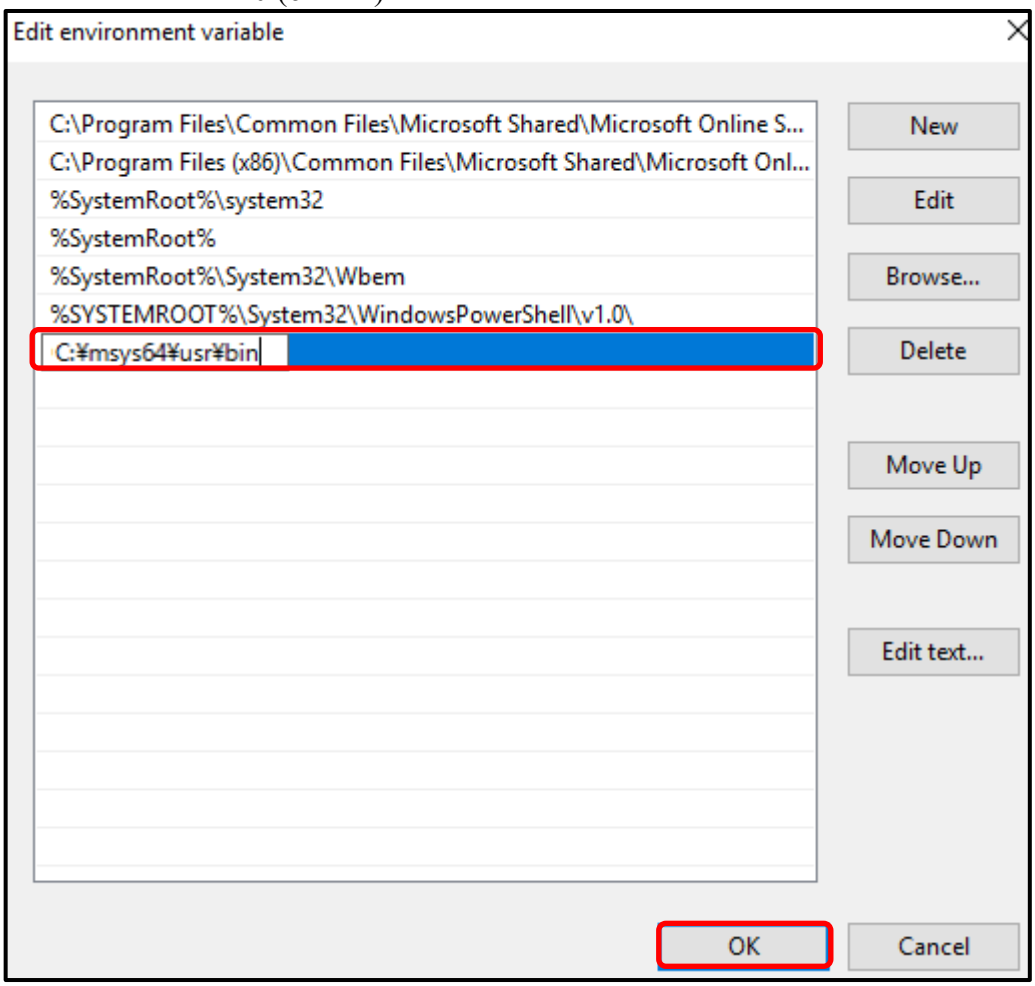


(5) Select the entry for [Path] in the [System variables] section, and click on the [Edit] button.



(6) Add "C:\msys\usr\bin" to the entries in the [Value] text box, and click on the [OK] button.
Note: If the 64-bit OS version is used for the client PC, the corresponding part is replaced with "C:\msys64\usr\bin".

■ For Windows 10 (64 bits)



(7) Close all windows of the [Control Panel] by clicking on the [OK] buttons.

2.1.3 Installing the Visual C++ Run-time Components

This subsection describes how to install the Visual C++ run-time components, which are required in writing files to the target board.

1. Download the installer from the website below.
   Because e2studio is a 32-bit application, select the 32-bit version installer instead of the 64-bit version.
   https://www.microsoft.com/en-us/download/details.aspx?id=48145

2. Double-click on the icon of the downloaded installer file to start the installer, and install the files according to the setup procedure.

2.1.4 Installing the Serial Driver

Install the serial driver according to the procedure given in the user's manual attached to the board used.

2.1.5 Installing the Plug-ins

2.1.5.1 Installing the Nebula Plugin

1) Start the e² studio and select [Help] -> [Install New Software] from the menu bar to open the [Install] dialog box.

   Click on the [Add] button, enter "Eclipse Nebula Plugin Update Site" as a name and "http://download.eclipse.org/nebula/releases/1.3.0/" as a location, and click on the [OK] button.

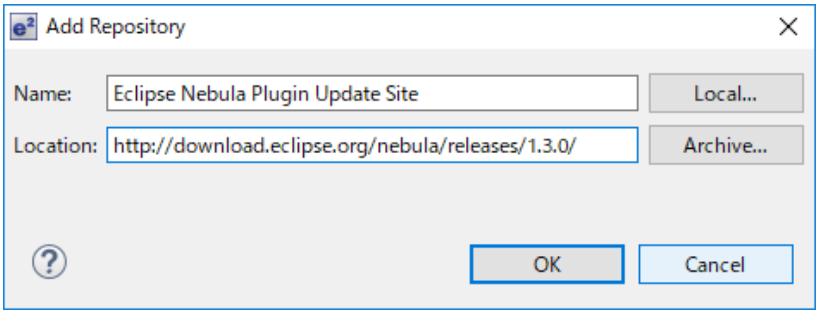


Figure 2-1 Setting Nebula Plugin Location

Select [Nebula Release all Widgets and Examples] -> [Nebula Widgets] from the list, click on the [Next] button, confirm the license, and install the plug-ins.

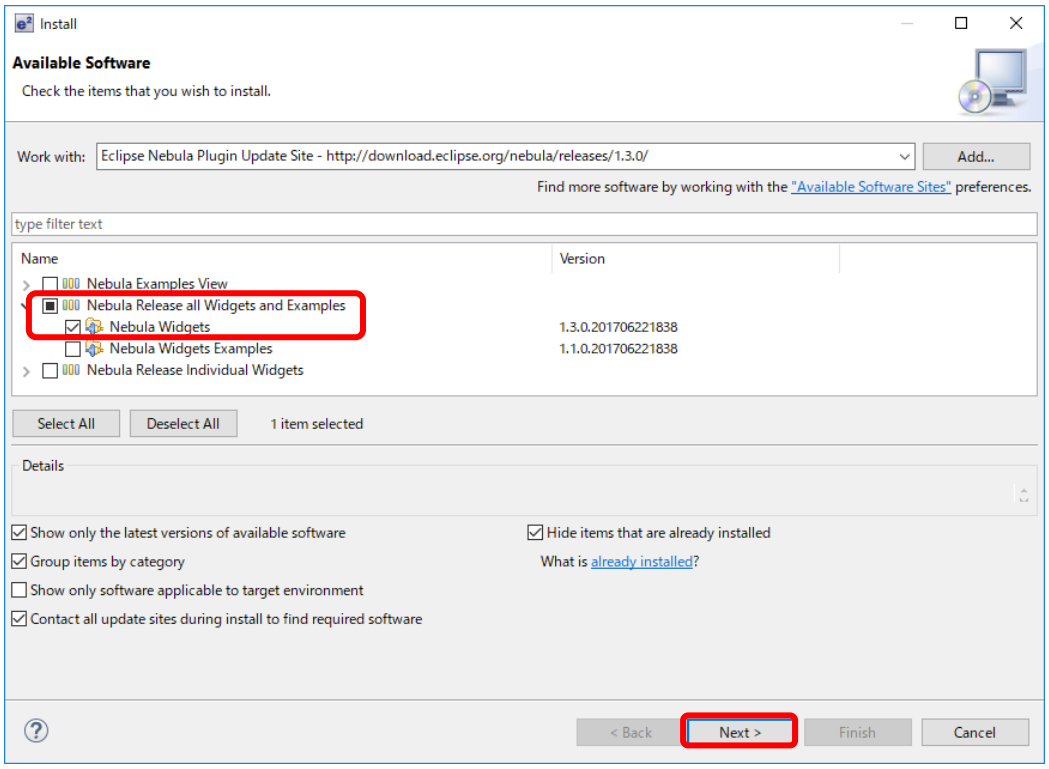


Figure 2-2 Selecting Nebula Widgets

2.1.5.2 Installing the New RZ/G Linux Platform Tool Plug-ins

This subsection describes the procedure for installing the new RZ/G Linux platform tool plug-ins.

Types of plug-ins:
There are three kinds of plugins.
Install the plug-in of the tool you use according to Procedure for installing new plug-ins.
Installation needs to be done for each plug-in.

- Plug-ins for the Linux Customization tool
  RZG-LinuxPF-Tools_ LinuxCustomization_～.zip
- Plug-ins for the Verification tool and Analysis tool
  RZG-LinuxPF-Tools_Verification&Analysis_～.zip
- Plug-ins for the Test programs
  RZG-LinuxPF-Tools_TestPrograms_～.zip

Note: If you use the Verification tool, you must install "Plug-ins for the Verification tool and Analysis tool" and "Plug-ins for the Test programs".

Procedure for installing new plug-ins:

1) Unzip the distributed ZIP file to the location immediately under the C drive.

2) Start the e2 studio and select [Help] -> [Install New Software] from the menu bar to open the [Install] dialog box.

3) Click on the [Add] button and then the [Archive] button, and select the "C:¥ RZG-LinuxPF-Tools¥PluginRepository¥com.renesas.rzglinuxpf.tools.～.zip" file in the folder having been unzipped in step 1) above.
   Notes: 1. If the ZIP file has been unzipped to the location other than immediately under the C drive, select the "com.renesas.rzglinuxpf.tools. ～.zip" file in the unzipped folder.

4) Select [RZ/G Linux Platform tools], click on the [Next] button, confirm the license and signature, and install the plug-ins.

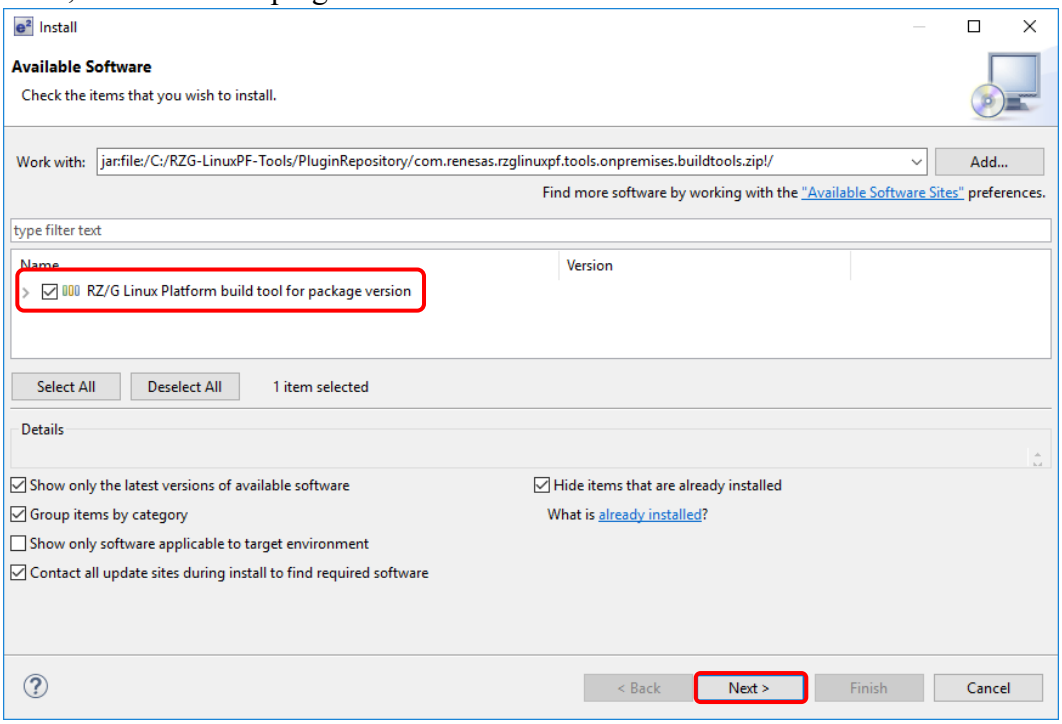


Figure 2-3 Selecting RZ/G Linux Platform Tools

2.1.5.3 Updating the RZ/G Linux Platform Tool Plug-ins

This subsection describes the procedure for updating the RZ/G Linux platform tool plug-ins.

Procedure for updating:

1) Please uninstall plug-ins already installed, according to the procedure in 2.2.1.

2) If e^2 studio is running, please exit it.

3) Replace the RZG-LinuxPF-Tools folder, which has been unzipped and created upon previous installation of plug-ins, with the contents of the updated version of the ZIP file.

4) Install the updated plug-ins, according to the procedure in 2.1.5.2.

   Note:
   After selecting plug-in on the procedure 3) in 2.1.5.2, if displayed like "There are no categorized items" and cannot install, change the folder name of "RZG-LinuxPF-Tools" to a different name and install again.

2.1.6 Creating a Folder for Storing Tool Data

Before using the verification tool, create the following folders.

1) Folder for storing the results of verification executed by the verification tool
   A specific folder is necessary for the verification tool to store the results of verification in its internal format. Be sure to create this folder and specify the path to it as one of items to be set for using the verification tool as listed in Table 4-2. If the folder is not present in the specified path, the tool attempts to create the folder automatically.

   Note that the following path will have been specified by default.
   <User folder under Windows>\.RZG-LinuxPF-Tools\Result

   Example when taking the user as user1: C:\Users\user1\.RZG-LinuxPF-Tools\Result

2) Folder for storing the user-created test programs
   When registering the user-created test program, a specific folder is necessary for the verification tool to store the user-created test program in the internal format. This folder is required when using the user-created test program with the verification tool.

   The folder can be created in any folder; however, specify the path to the created folder as one of items to be set in Table 4-2.

3) Folder for storing the test program for the software add-on when this has been imported
   When registering the test program distributed with the software add-on, a specific folder is necessary for the verification tool to store the test program in its internal format.

   If the folder is not present in the specified path, the tool attempts to create the folder automatically.

   Note that the following path will have been specified by default.
   <User folder under Windows>\.RZG-LinuxPF-Tools\AddonTP

   Example when taking the user as user1: C:\Users\user1\.RZG-LinuxPF-Tools\AddonTP

4) Folder for edit parameter for test program
This is a folder for the Verification Tool to store this parameter (see section 4.10) in internal format when edited the parameters that can be set for the test program.
When editing the parameters for test program using the Verification Tool, you must specify the path of the folder as listed in Table 4-2.
If the folder is not present in the specified path, the tool attempts to create the folder automatically.

Note that the following path will have been specified by default.
<User folder under Windows>\.RZG-LinuxPF-Tools\EditData

Example when taking the user as user1: C:\Users\user1\.RZG-LinuxPF-Tools\EditData

2.2 Uninstallation Method

2.2.1 Uninstalling the Plug-ins

Select [About e² studio] on the [Help] menu on the e² studio, and select [Installation Detail] button placed on lower part for the [About e² studio] dialog, then the screen shown in Figure 2-4 is displayed.
On the screen, select [RZ/G LinuxPF~] and click on the [Uninstall] button, and the execution window for uninstallation shown in Figure 2-5 is displayed. Click on the [Finish] button.

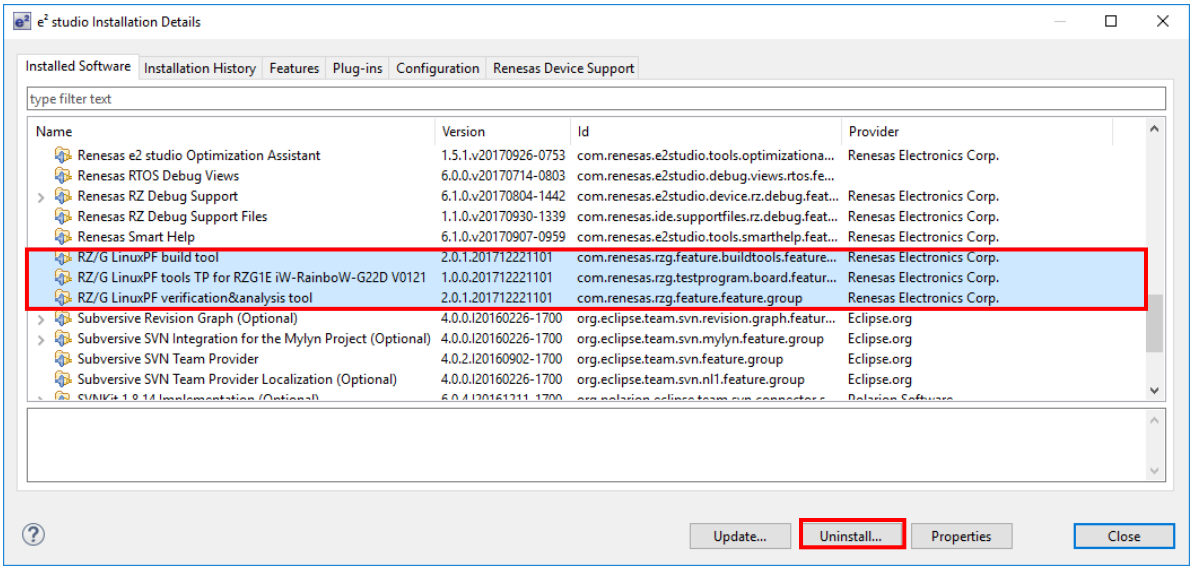


Figure 2-4 Screen of Installation Details

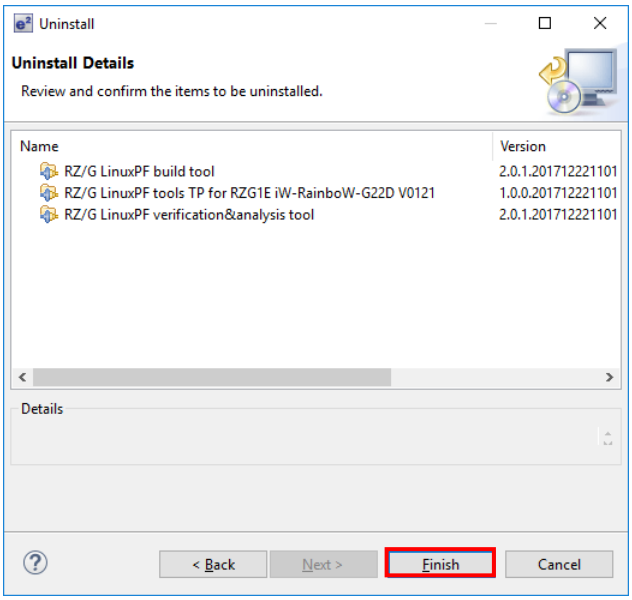


Figure 2-5 Screen of Uninstallation Details

2.3 Confirming the Installed Plug-in Version

Select [About e^2 studio] on the [Help] menu on the e^2 studio, and select [Installation Detail] button placed on lower part for the [About e^2 studio] dialog. On the [e2 studio Installation Details] screen displayed, select the [Installed Software] tab.

You can confirm the version of [RZ/G LinuxPF~] displayed in the [Version] column of the list as shown in Figure 2-6.

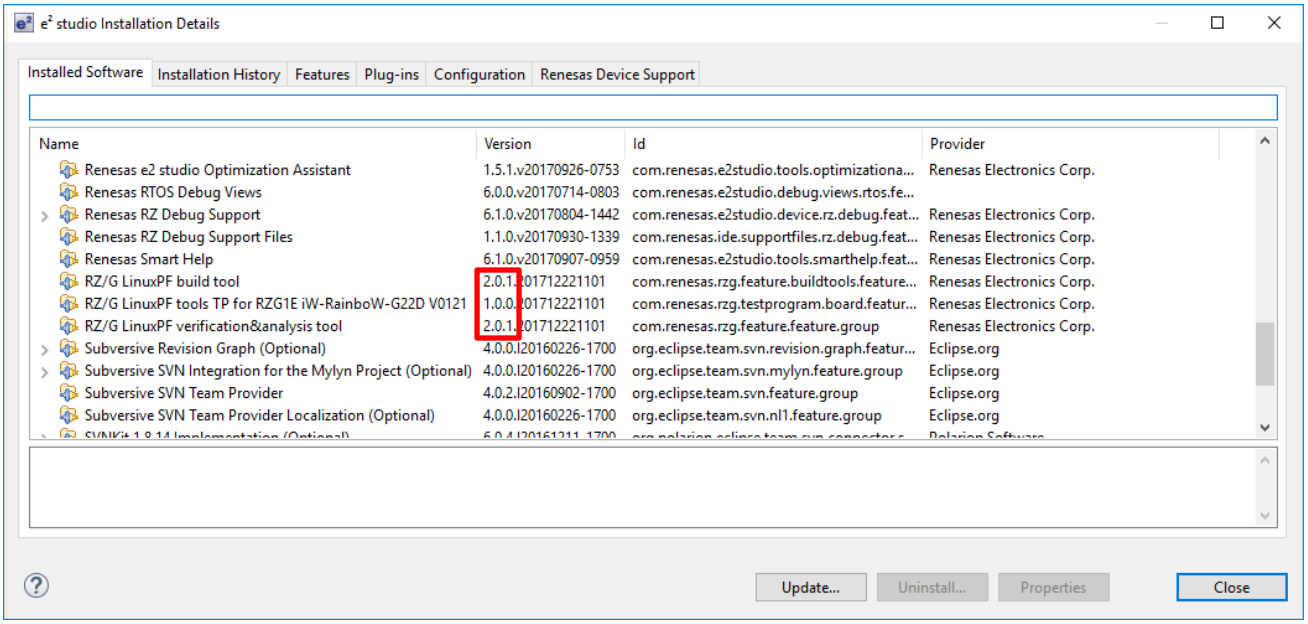


Figure 2-6 Example of Confirming the Plug-in Version

3. Handling the Linux Customization Tool

This section describes how to handle the Linux Customization tool.

3.1 List of Functions

Common functions		
Category	**Function Name**	**Description**
Handling build server	User authentication	Authenticates users against the build server using the SSH key.
	Starting up the build server	Starts up the build server.
Creating a project	Embedded Linux project	Creates a project for build and target board control.
	Embedded Linux custom BSP project	Creates a project for customizing the BSP.
	Deleting a project	Deletes the embedded Linux project or embedded Linux custom BSP project that has been created.

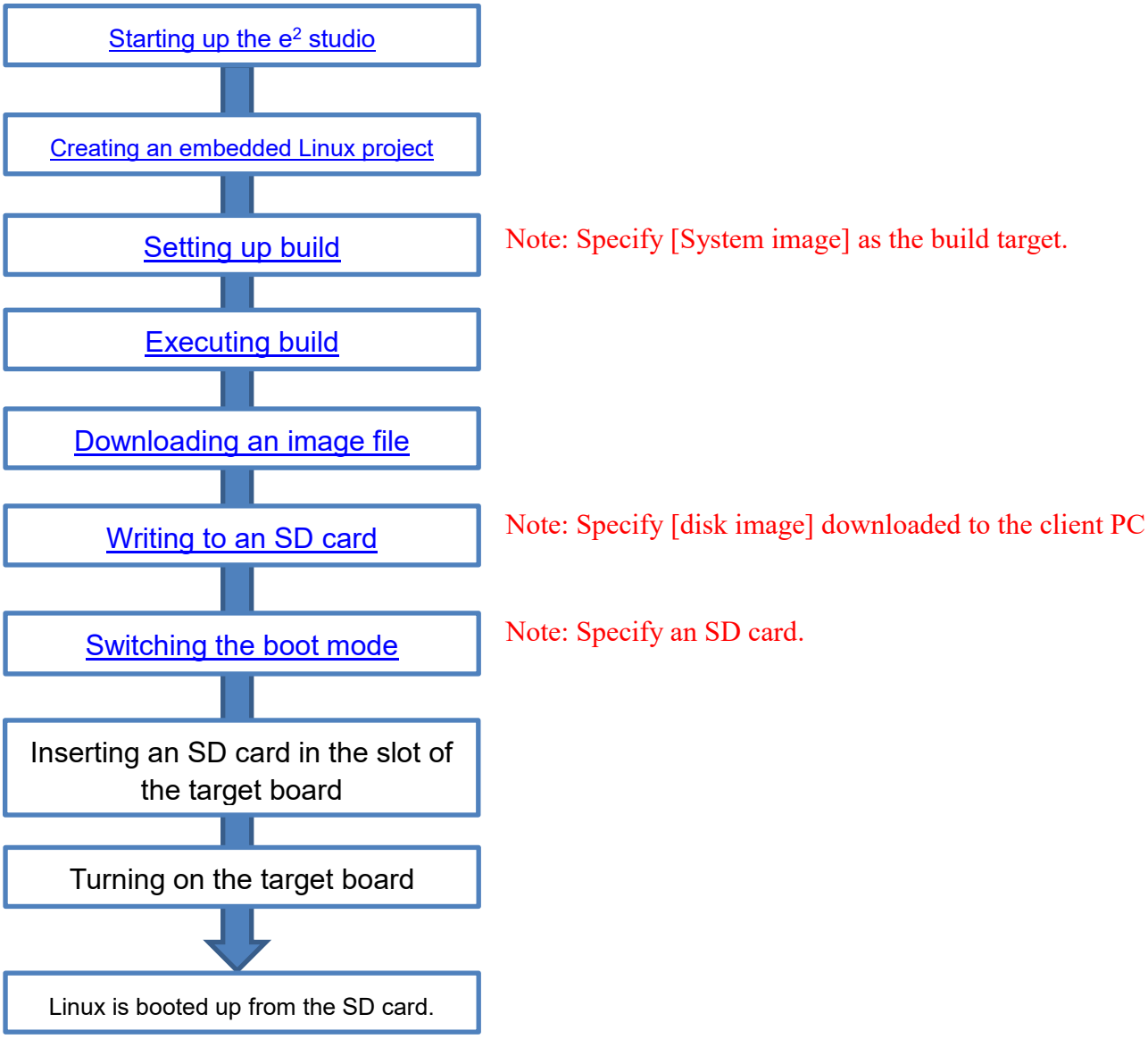
Functions for an embedded Linux project		
Category	**Function Name**	**Description**
Adding	User program	Adds a program created by a user as the build target.
Building	Setting up build	Selects the build target, build target BSP, and build target components.
	Executing build	Executes the build process according to the setup contents.
Downloading	Results of building	Downloads the built image file, executable binary file, and object to the client PC. • Disk image • eMMC installer • User program
	U-Boot	Downloads the boot loader contained in the BSP.
Writing	Image file	Writes the image files downloaded to the client PC to the specified SD card or an eMMC.
	User program	Writes the executable user program file downloaded to the client PC to the execution environment image file stored in an eMMC on the board or an SD card.
	Boot mode	Switches the device (an SD card or an eMMC on the board) for booting (starting up) Linux.

Functions for an embedded Linux custom BSP project		
Category	**Function Name**	**Description**
Customizing Yocto recipes	Editing Yocto recipes	Edits Yocto recipes.
	Adding Yocto recipe patches	Adds Yocto recipes.
Customizing the kernel	Changing the kernel configuration	Changes the kernel configuration.
	Initializing the kernel configuration	Initializes the kernel configuration.
	Editing the kernel code	Edits the kernel code.
	Removing the contents of editing the kernel code	Removes the contents of editing the kernel code.
Customizing U-Boot	Editing the U-Boot code	Edits the U-Boot code.
	Removing the contents of editing the U-Boot code	Removes the contents of editing the U-Boot code.
Git	Registering with the Git	Registers the contents of customization of the Yocto recipes, kernel, and U-Boot with the Git.

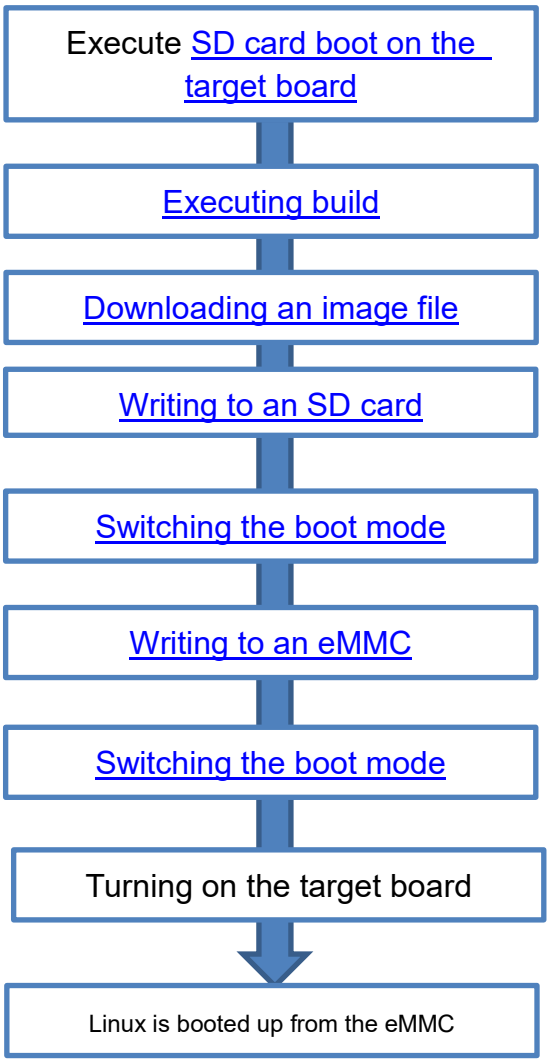
3.2 Application Examples

The following shows application examples of the Linux Customization tool.

3.2.1 SD Card Boot on the Target Board



3.2.2 eMMC Boot on the Target Board



Note: Specify [eMMC installer image] as the build target.

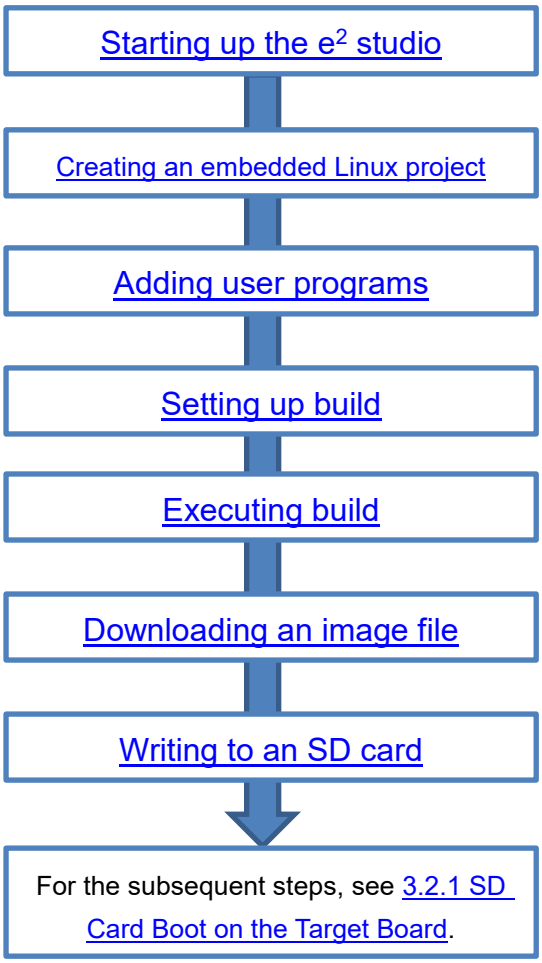
Note: Specify [eMMC installer image] downloaded to the client PC.

Note: Specify an SD card.

Note: The eMMC installer is automatically written to an eMMC.

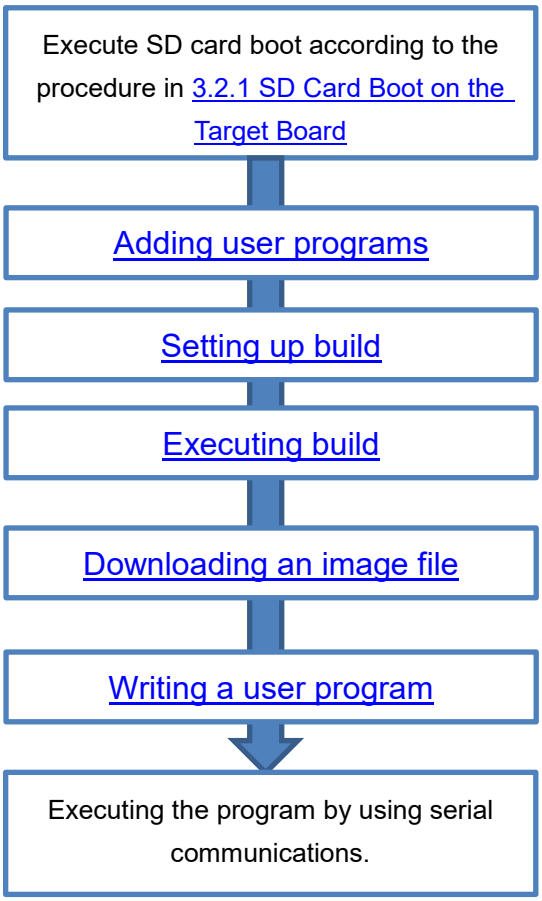
Note: Specify the eMMC.

3.2.3 Combining User Program Build with SD Card Boot



Note: Specify [System image] as the build target.

3.2.4 Directly Writing a User Program to the Target Board

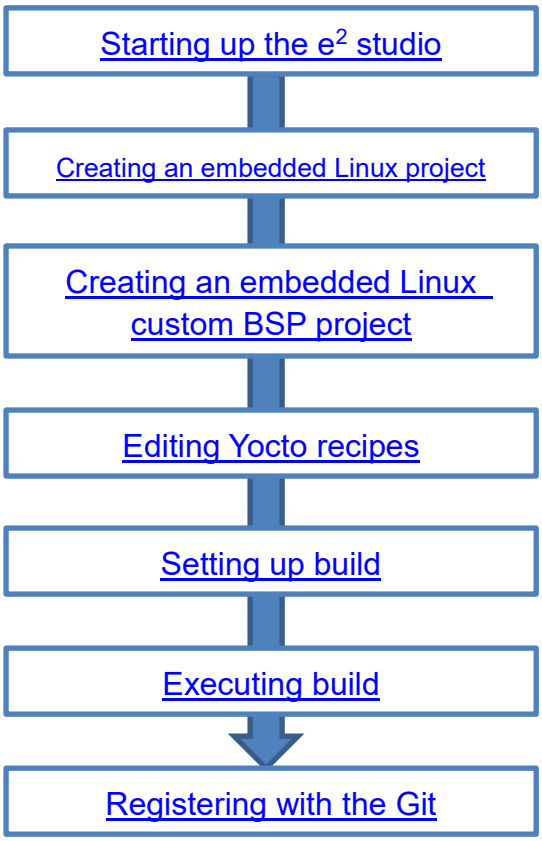


Note: Specify [User programs] as the build target.

Note: Specify [User program: executable binary].

Note: Automatic program execution immediately after booting is also possible by setting shell script in start_userapp.sh

3.2.5 Customizing a Yocto Recipe and Registering the Results of Customization with the Git Repository



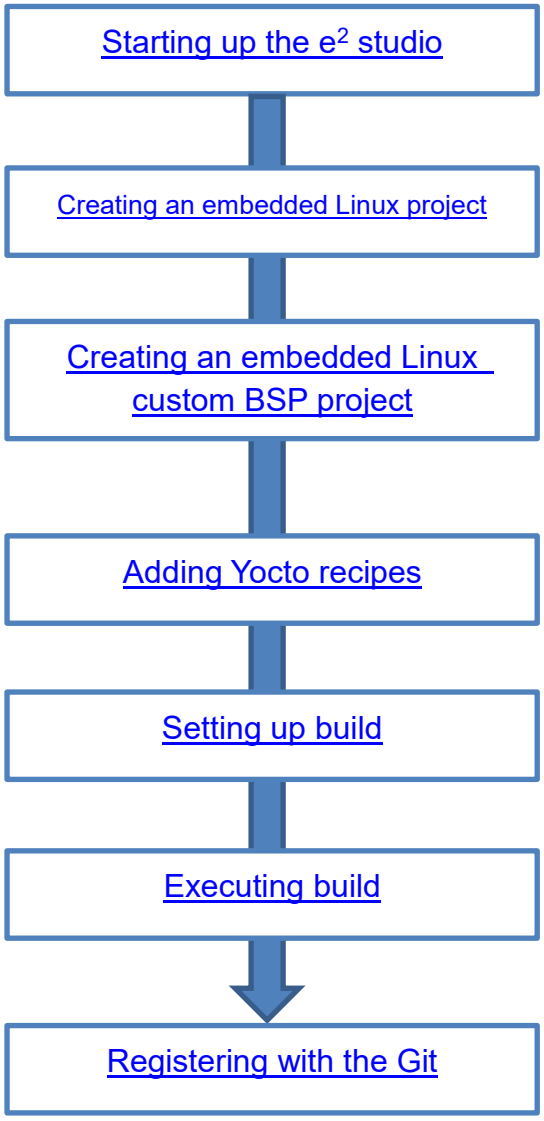
Note: A Yocto recipe is downloaded to the client PC.

Note: Specify [Custom BSP project on the workspace] as the BSP.

Note: Build the customized BSP.

Note: Register the results of customization with the Git repository.

3.2.6 Adding a Yocto Recipe and Registering the Added Yocto Recipe with the Git Repository



Note: A Yocto recipe is downloaded to the client PC.

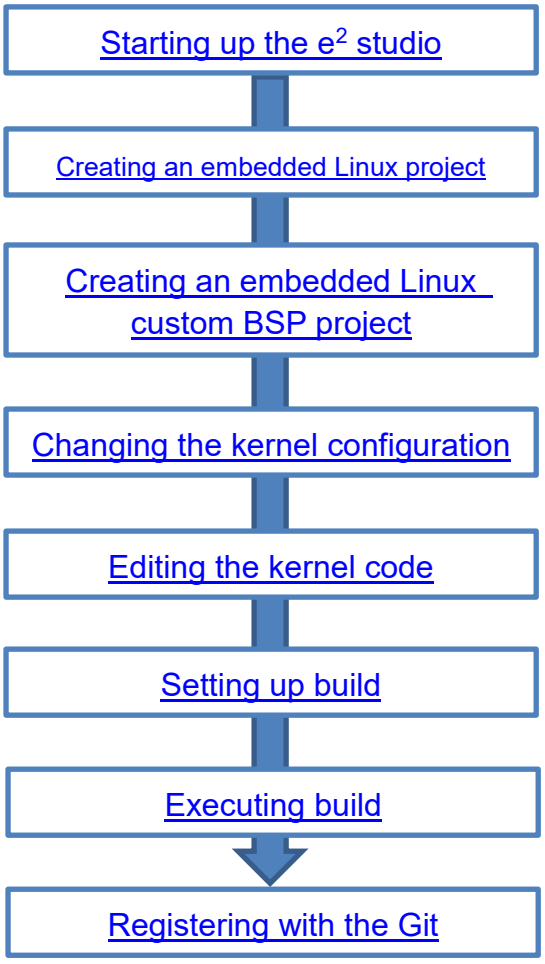
Note: Add the Yocto recipe which you have prepared on the client PC to the embedded Linux custom BSP project.

Notes: 1. Specify [Custom BSP project on the workspace].
2. Add the name of the recipe on the [Optional components] tabbed page.

Note: Build the BSP to which you have added the Yocto recipe.

Note: After confirming the operation of the tool, register the added Yocto recipe with the Git repository.

3.2.7 Customizing the Kernel and Registering the Results of Customization with the Git Repository



Note: Change the kernel configuration as necessary.

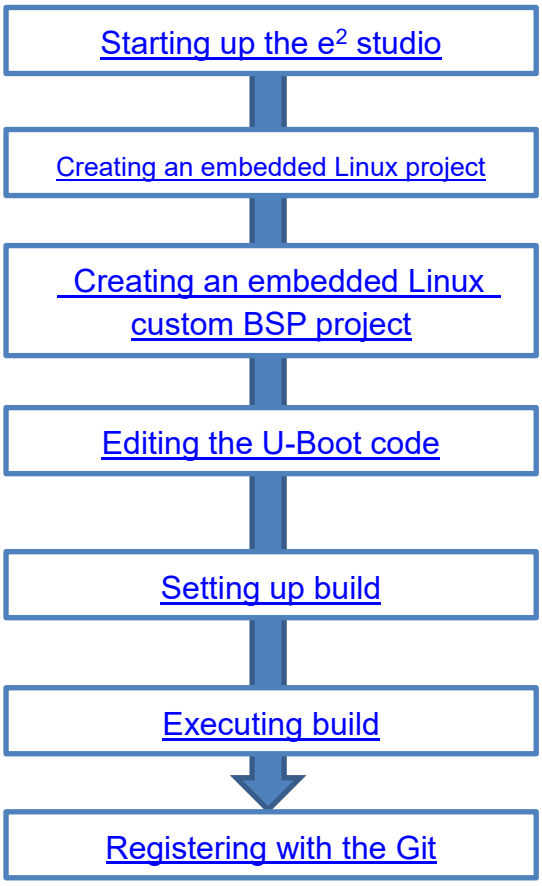
Note: The kernel source code is downloaded to the client PC.

Note: Specify [Custom BSP project on the workspace] as the BSP.

Note: Build the customized BSP.

Note: Register the results of customization with the Git repository.

3.2.8 Customizing U-Boot and Registering the Results of Customization with the Git Repository



Note: The U-Boot source code is downloaded to the client PC.

Note: Specify [Custom BSP project on the workspace] as the BSP.

Note: Build the customized BSP.

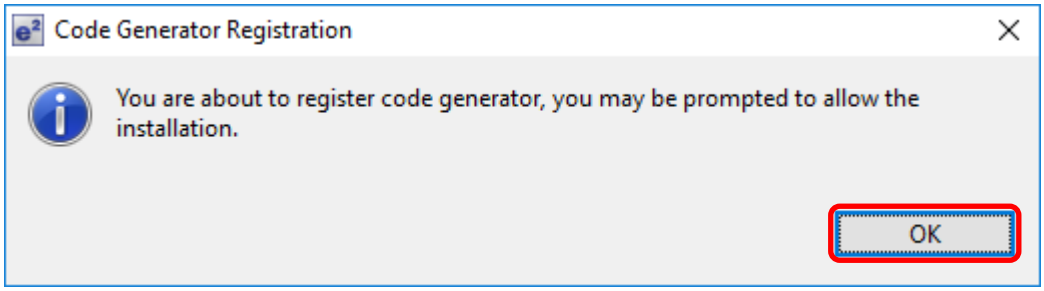
Note: Register the results of customization with the Git repository.

3.3 Starting Up the e² studio

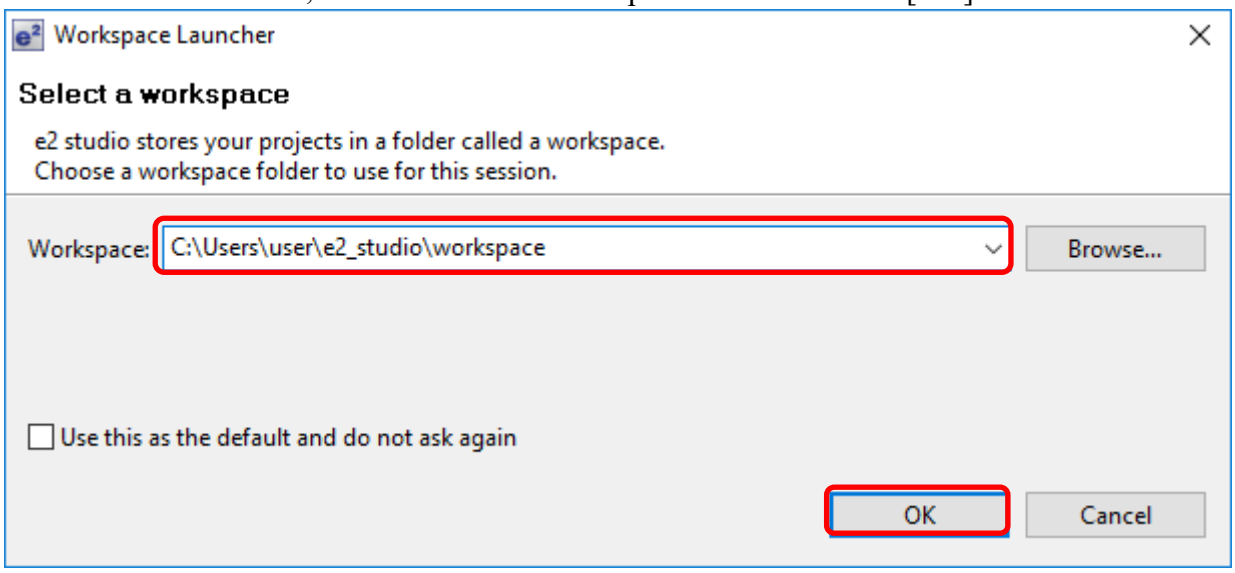
1. Start up the e² studio by using the start menu in Windows or by another method.

[Note 1 at Startup]

When the following [Code Generator Registration] message is displayed immediately after startup, click on the [OK] button.

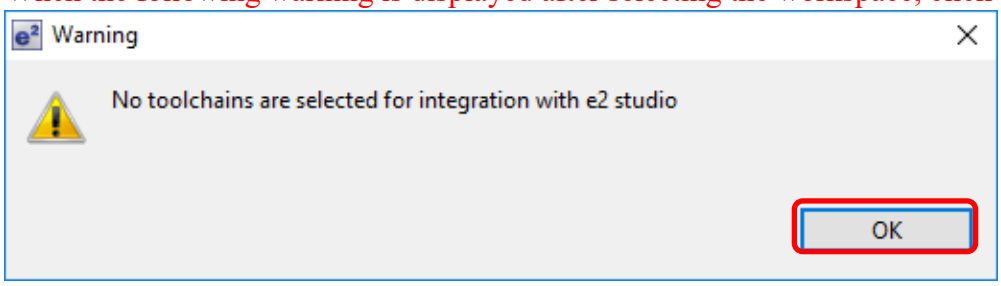


2. In the window below, select the desired workspace and click on the [OK] button.



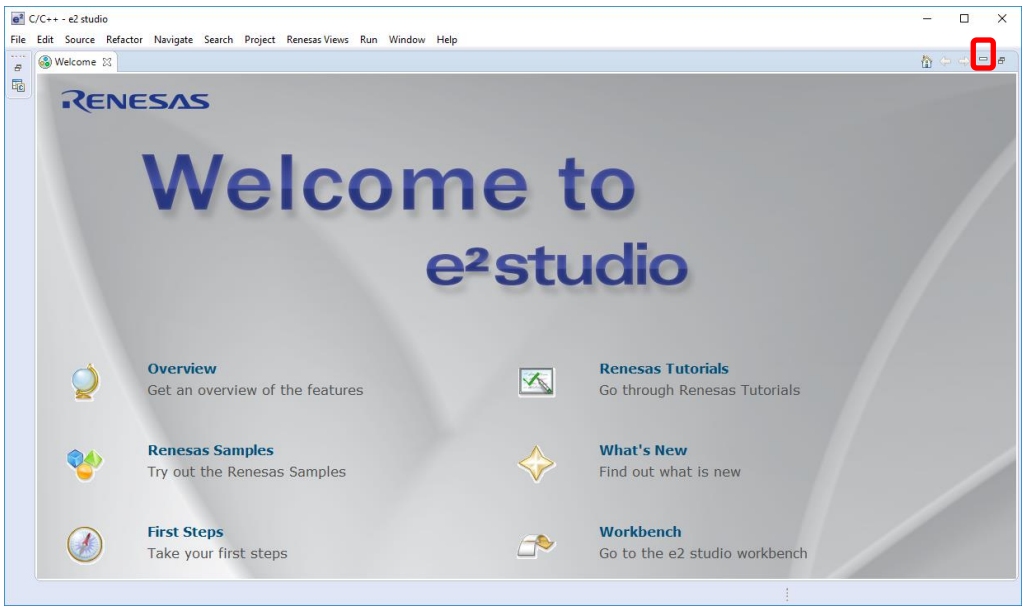
[Note 2 at Startup]

When the following warning is displayed after selecting the workspace, click on the [OK] button.



[Note 3 at Startup]

When the following window is displayed after selecting the workspace, select the minimize button from the upper right buttons.



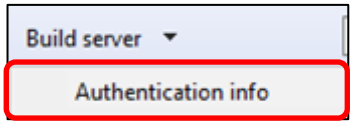
3.4 Handling Build Server

This section describes the user authentication of the build system in the Build server.

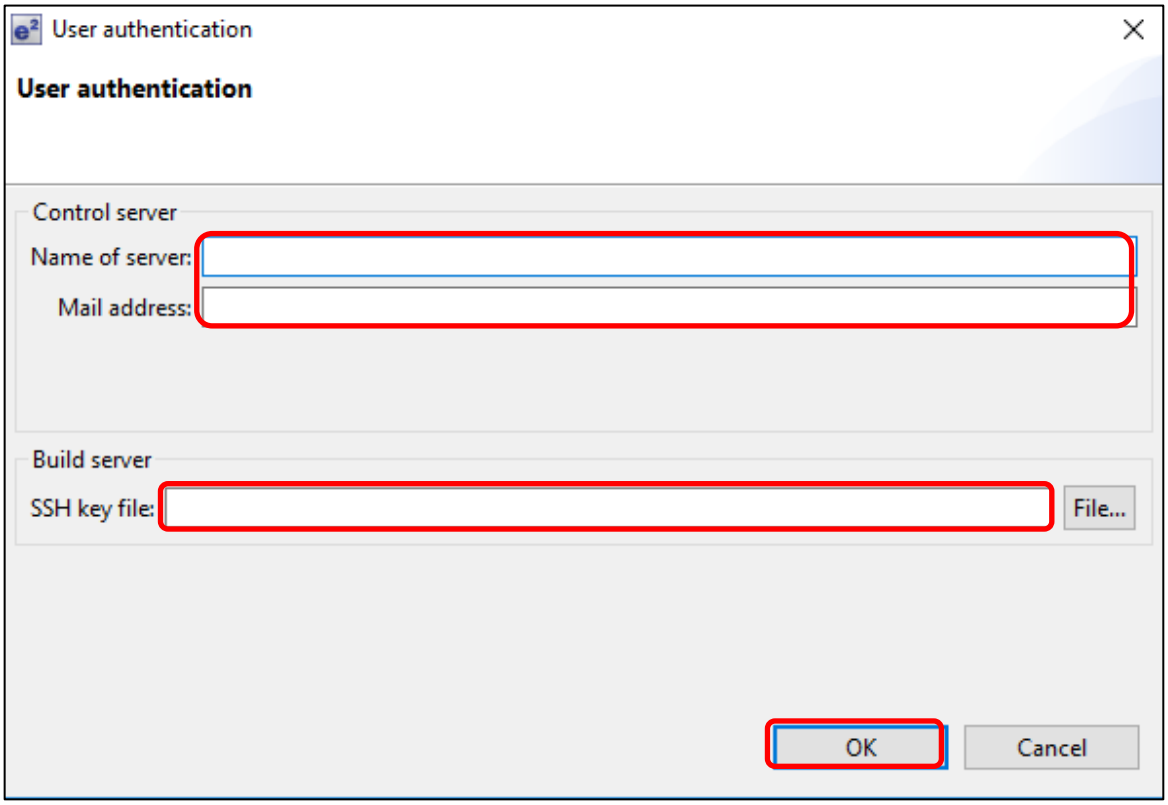
3.4.1 User Authentication

This subsection describes how to authenticate users against the build system in the Build server.

1. Select [Authentication info] from the [Build server] pull-down menu in the e^2 studio top screen.



2. The [User authentication] window shown below opens. Set the parameters required for authentication.



Parameter	Description
Name of server	Set the IP address of the Build server. Example: 192.168.149.134
Mail address	Set the email address of the user registered on the Build server.
SSH key file	Specify the file which was downloaded as SSH key from Build server.

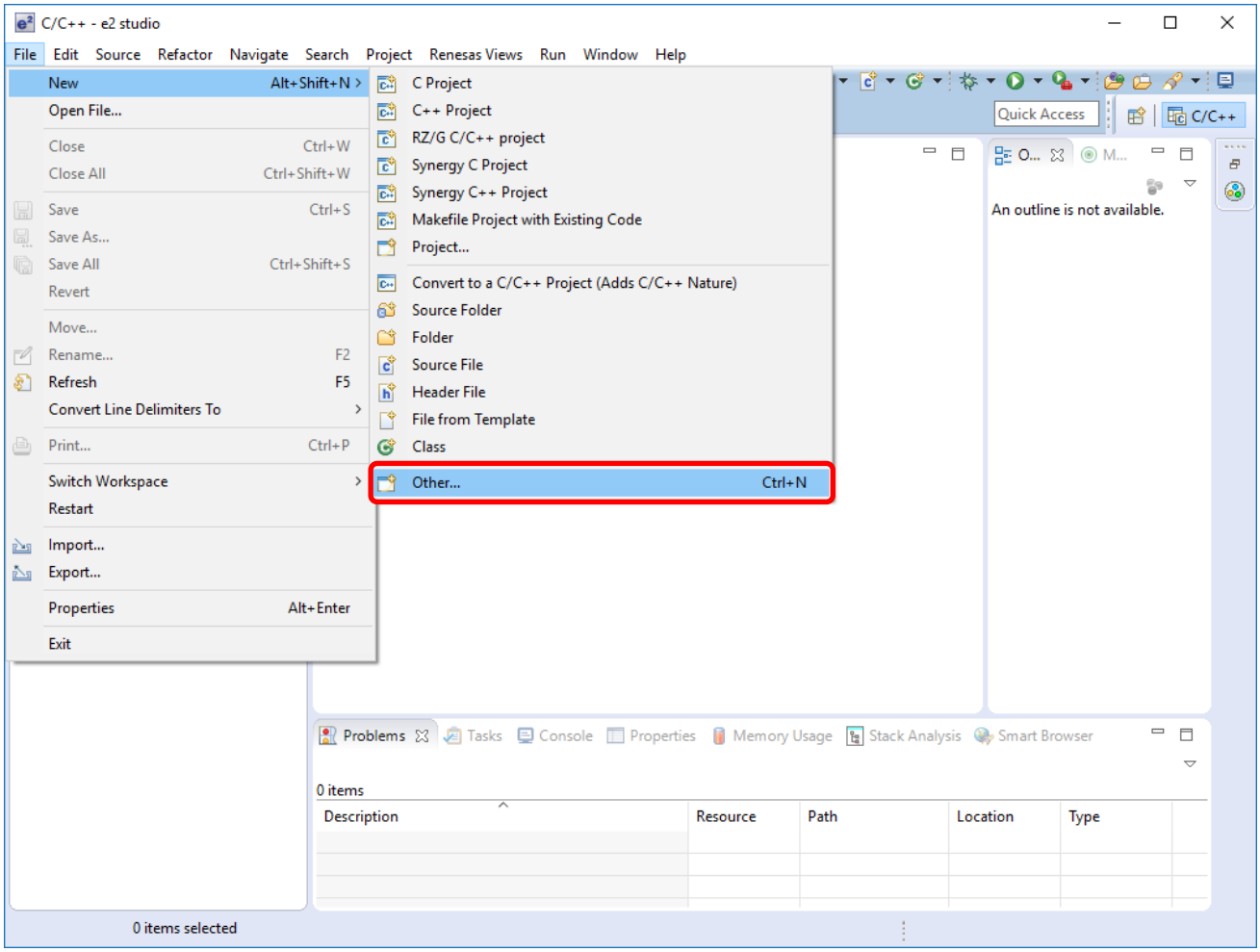
3.5 Creating a Project

This section describes the procedure for creating the project specific for the e[2] studio.

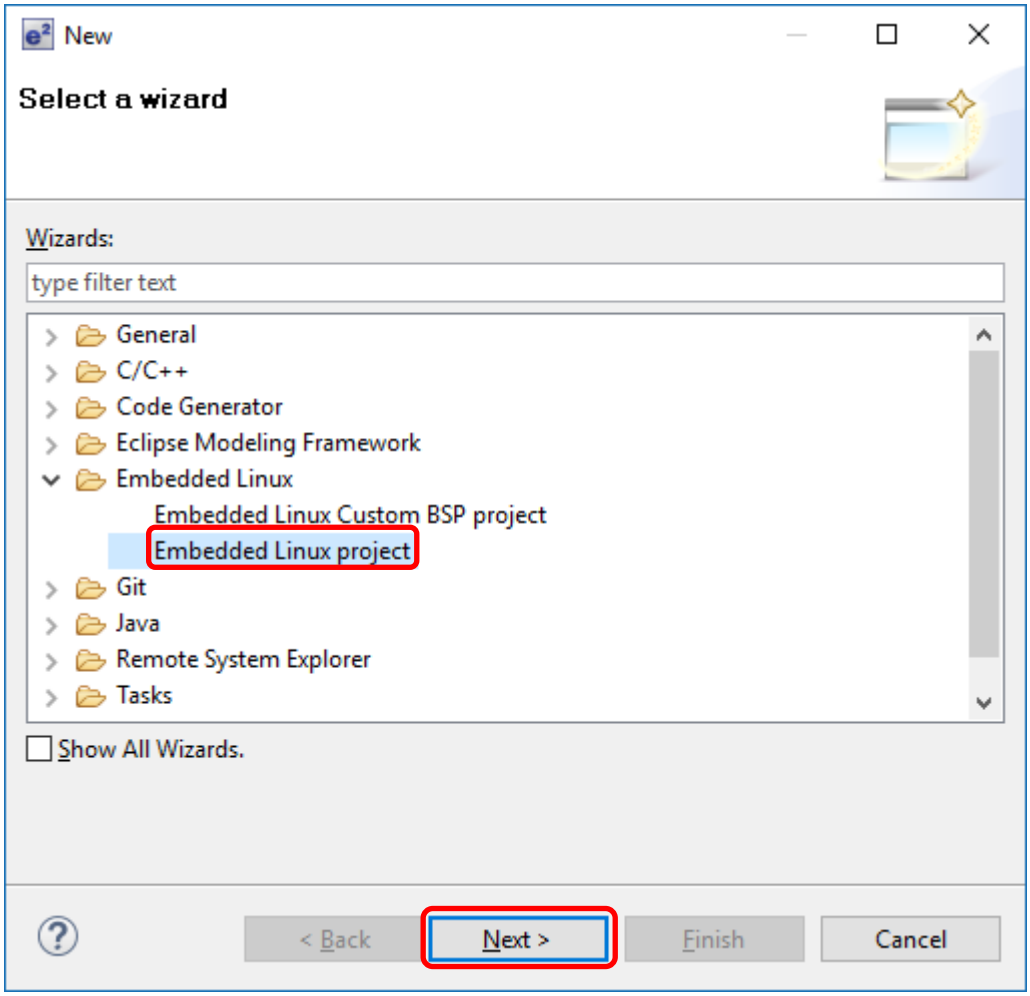
3.5.1 Creating an Embedded Linux Project

This subsection describes how to create an embedded Linux project to start up the target board.

1. Select [New] -> [Other] from [File] on the menu bar at the top of the window or from the icons on the menu bar at the top of the window.

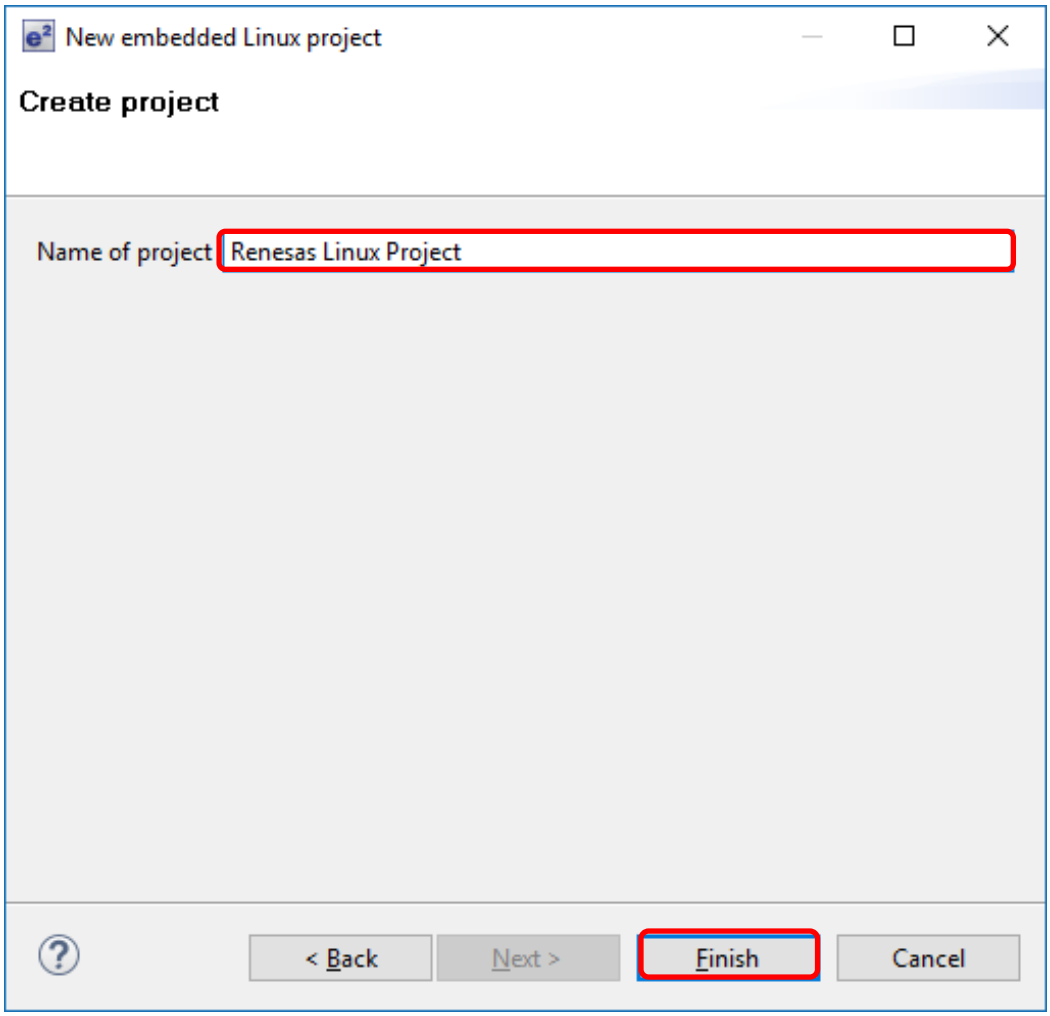


2. The [Select a wizard] window opens. Select [Embedded Linux project] under the [Embedded Linux] node, and click on the [Next] button.



3. The [User authentication] window opens. Set the parameters required for authentication. For details on the authentication procedure, see 3.4.1 User Authentication.
   Note: If authentication has already been made, the [User authentication] window does not open.

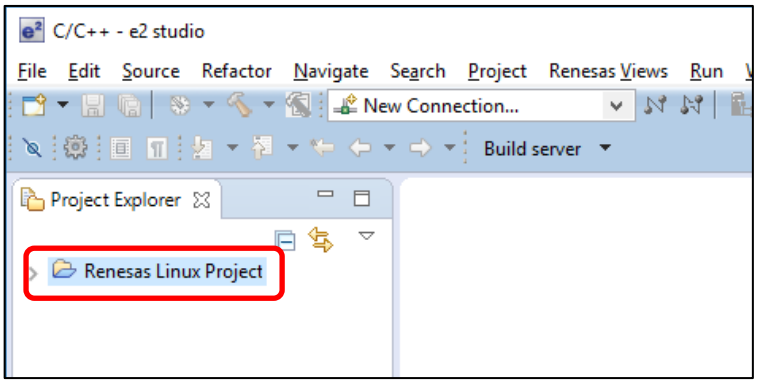
4. The [Create project] window opens. Set the following parameters.



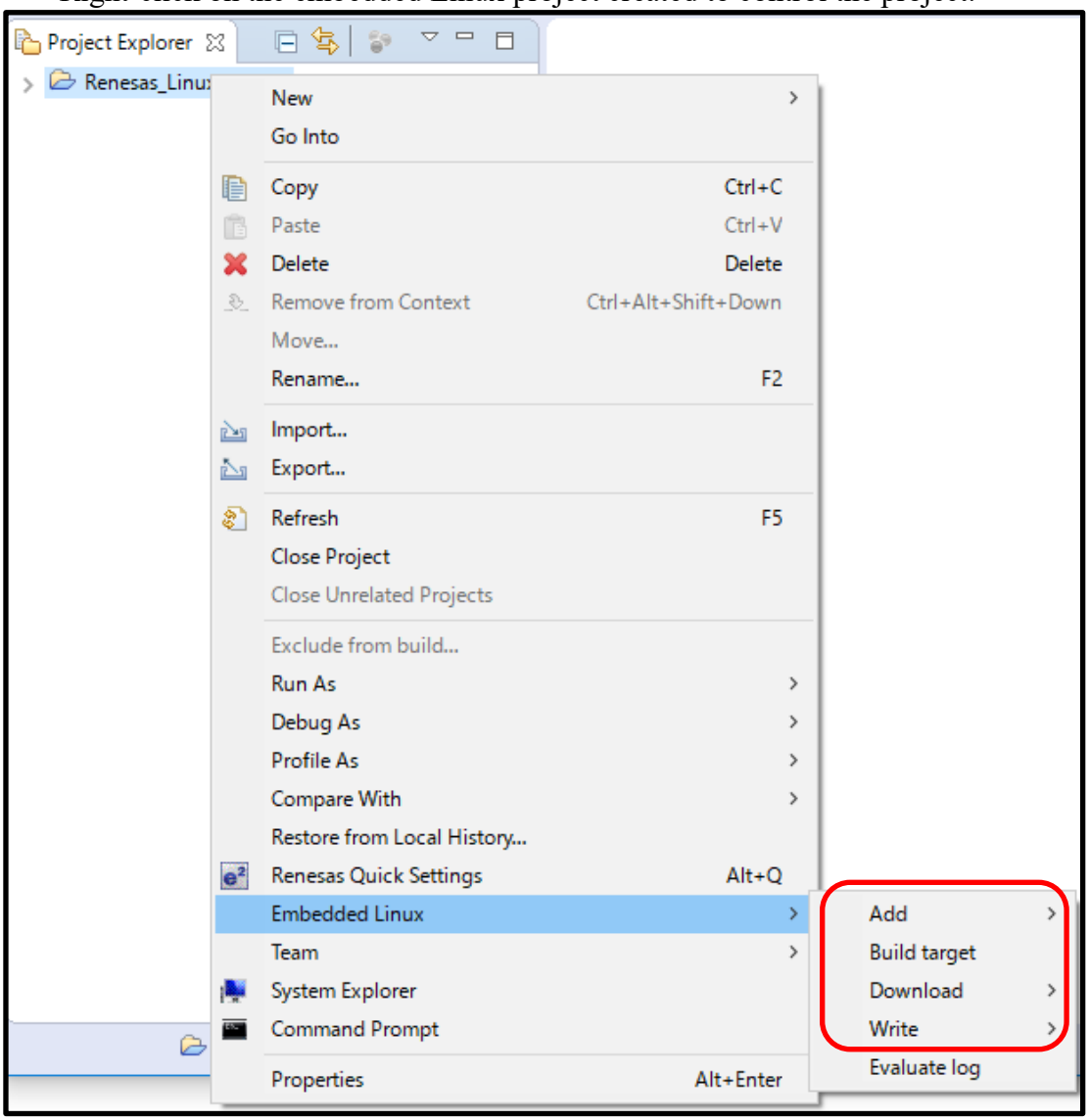
Parameter	Description
Name of project	Set the desired project name.

When parameter setting is completed, click on the [Finish] button.

5. A project is created and displayed on the [Project Explorer] pane.



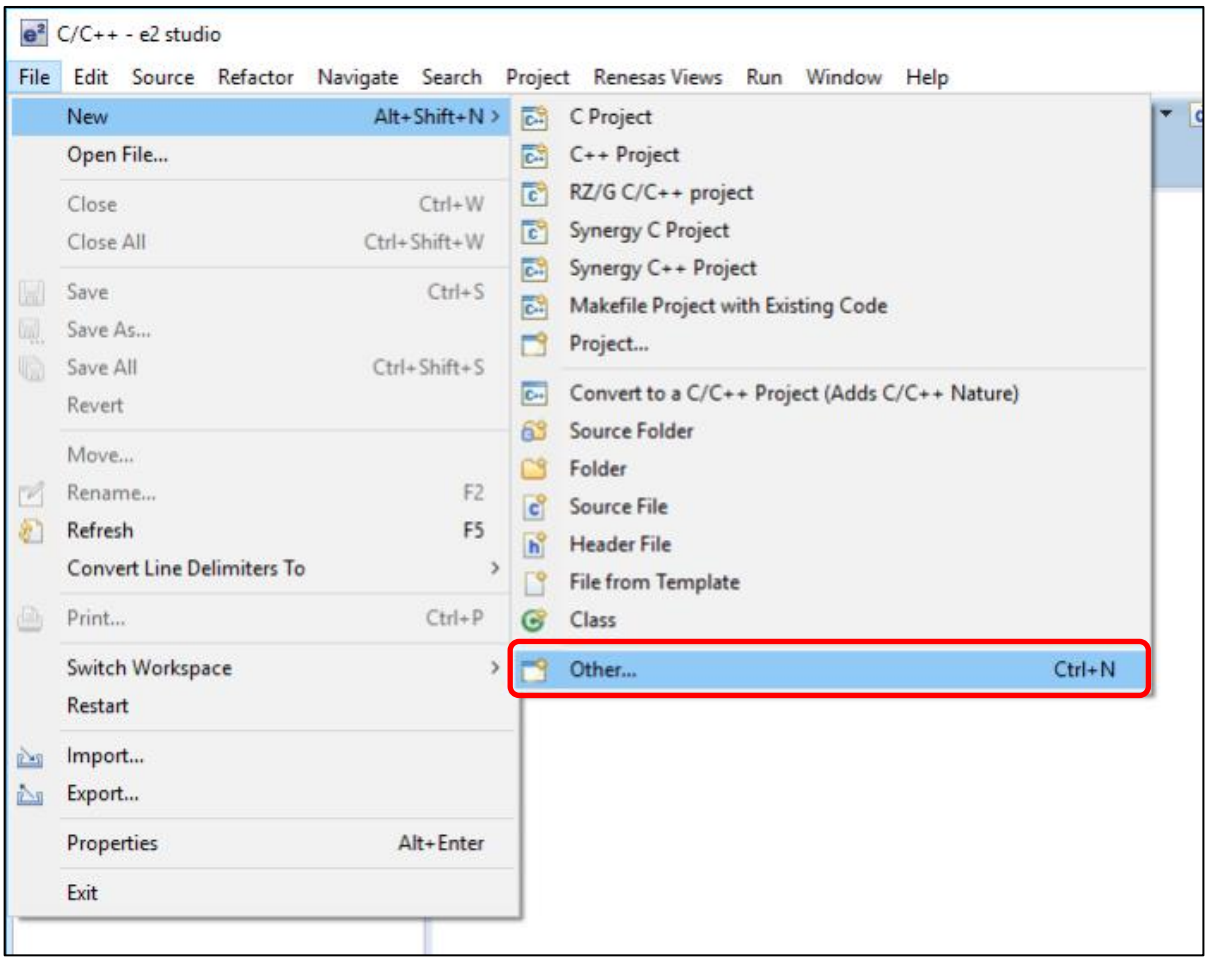
Right-click on the embedded Linux project created to control the project.



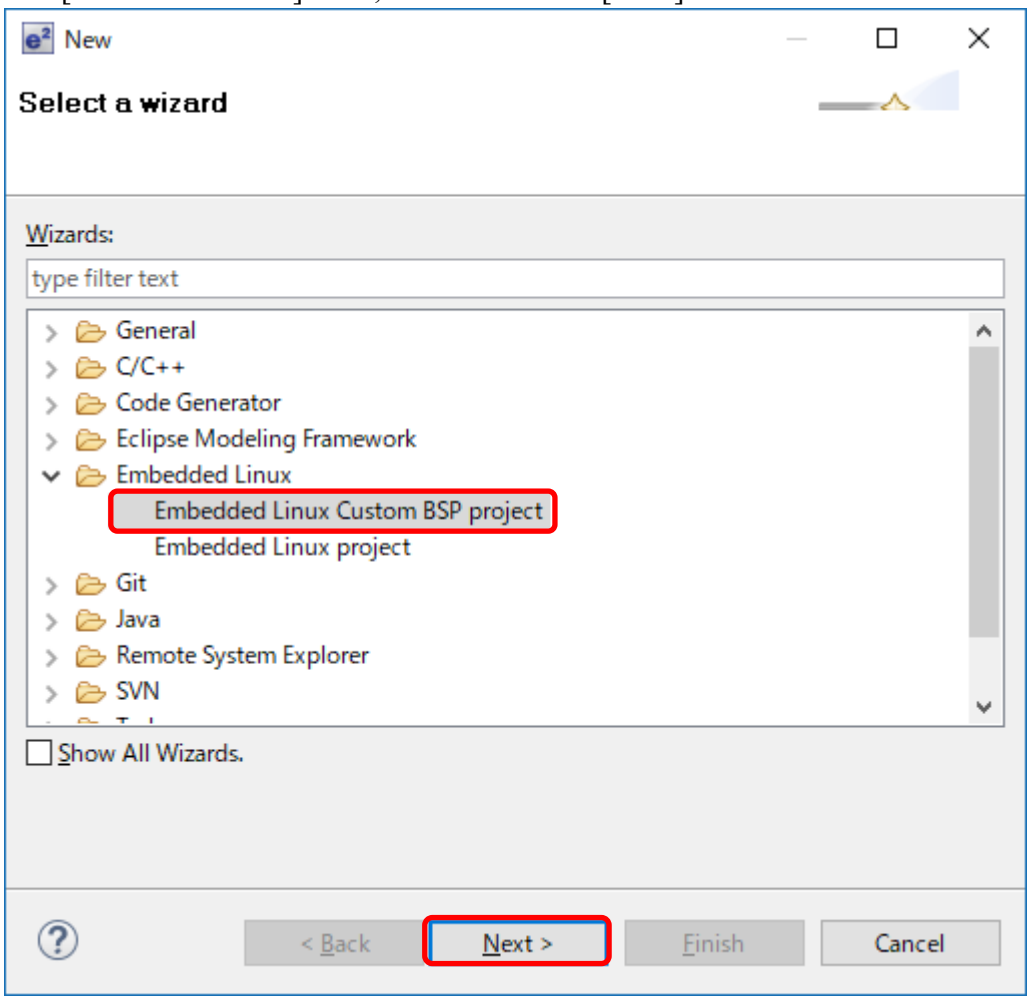
3.5.2 Creating an Embedded Linux Custom BSP Project

This subsection describes how to create an embedded Linux custom BSP project to customize the verified Linux package.

1. Select [New] -> [Other] from [File] on the menu bar at the top of the window or from the icons on the menu bar at the top of the window

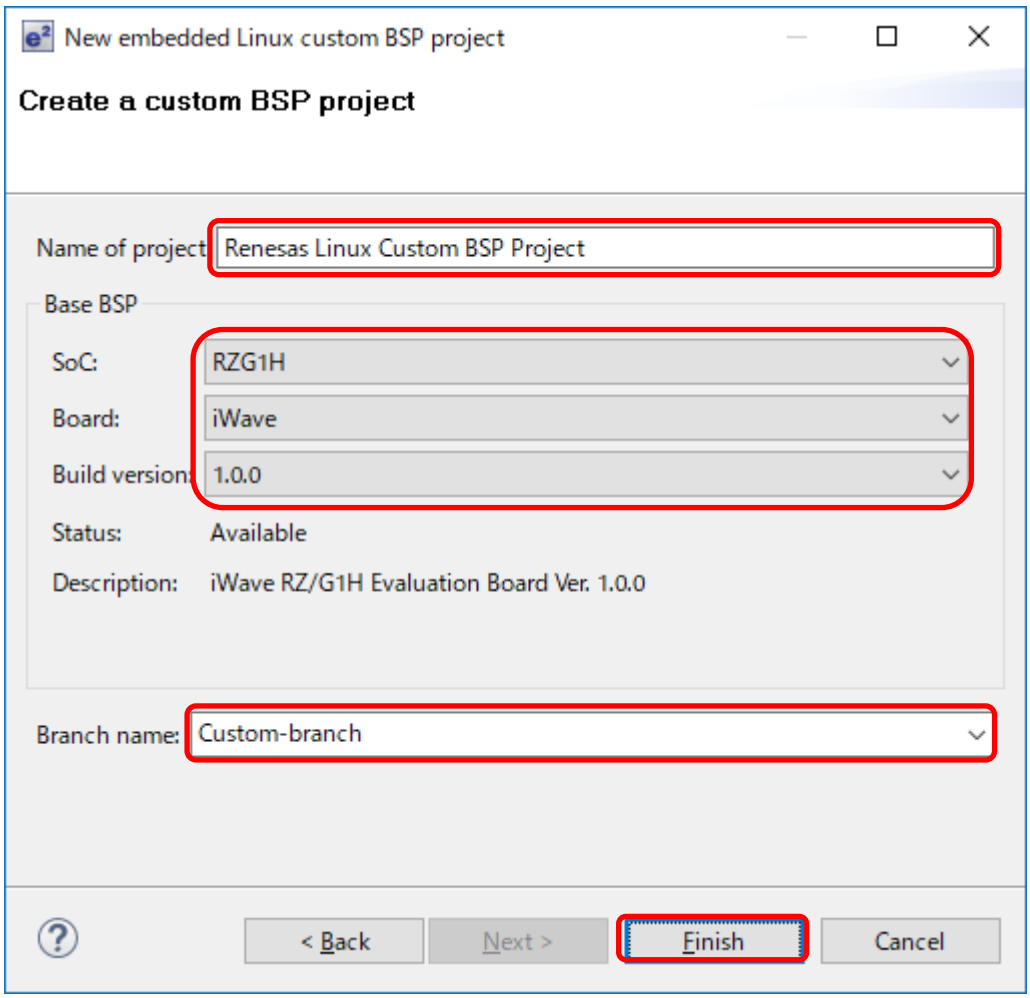


2. The [Select a wizard] window opens. Select [Embedded Linux Custom BSP project] under the [Embedded Linux] node, and click on the [Next] button.



3. The [User authentication] window opens. Set the parameters required for authentication. For details on the authentication procedure, see 3.4.1 User Authentication.
   Note: If authentication has already been made, the [User authentication] window does not open.

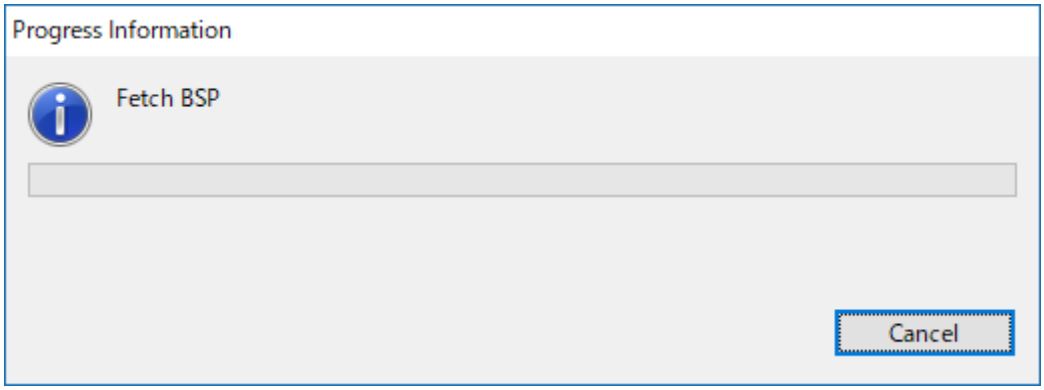
4. The [Create a custom BSP Project] window opens. Set the following parameters.



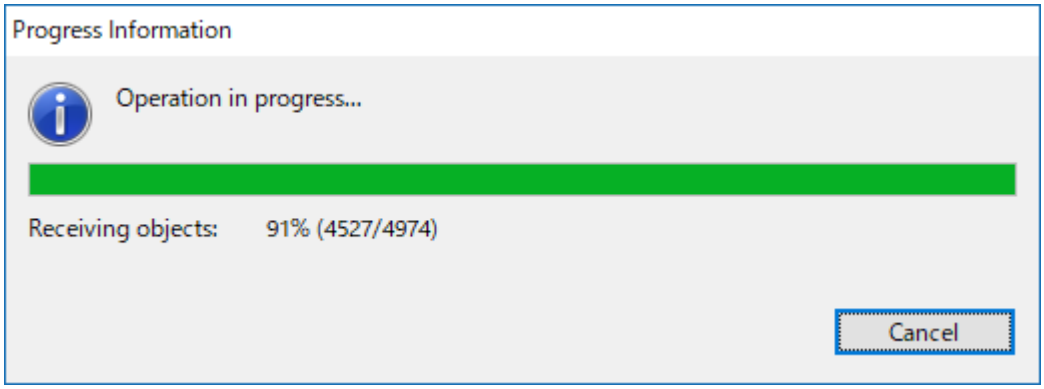
Parameter	Description
Name of project	Set the desired project name.
SoC	Specify the name of the chip on the target board to start up.
Board	Specify the name of the target board to start up.
Build version	Specify the BSP version number used to start up.
Branch name	Specify the name of the branch for control of the custom BSP recipes.

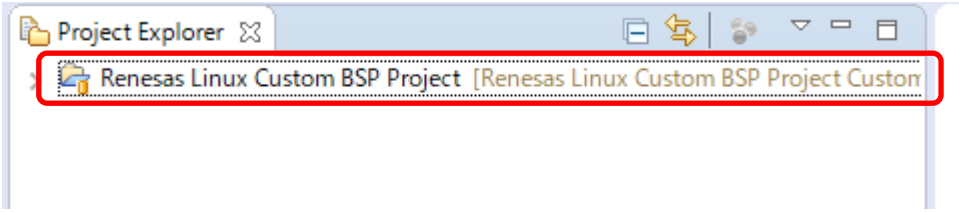
When parameter setting is completed, click on the [Finish] button.

5. The following window opens and downloading of the Yocto recipes from the build server to the client PC starts.
   For how to edit the downloaded Yocto recipes, see 3.11 Customizing Yocto Recipes.

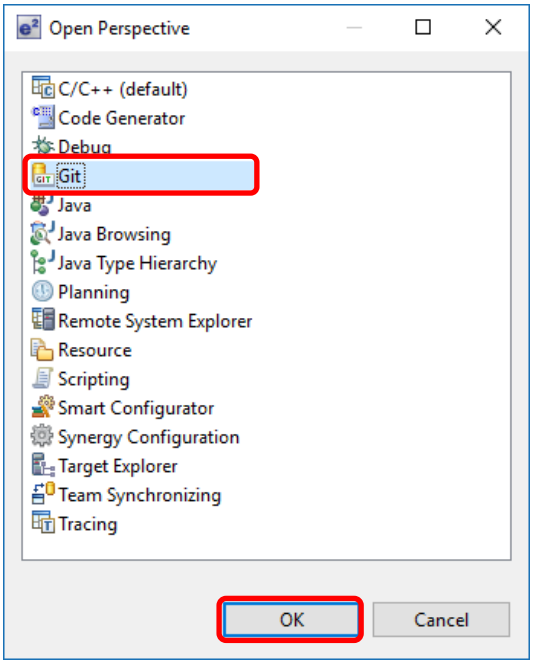


6. When the indicator reaches 100%, the creation of the project is finished, and you will see the project name displayed on the [Project Explorer] pane.

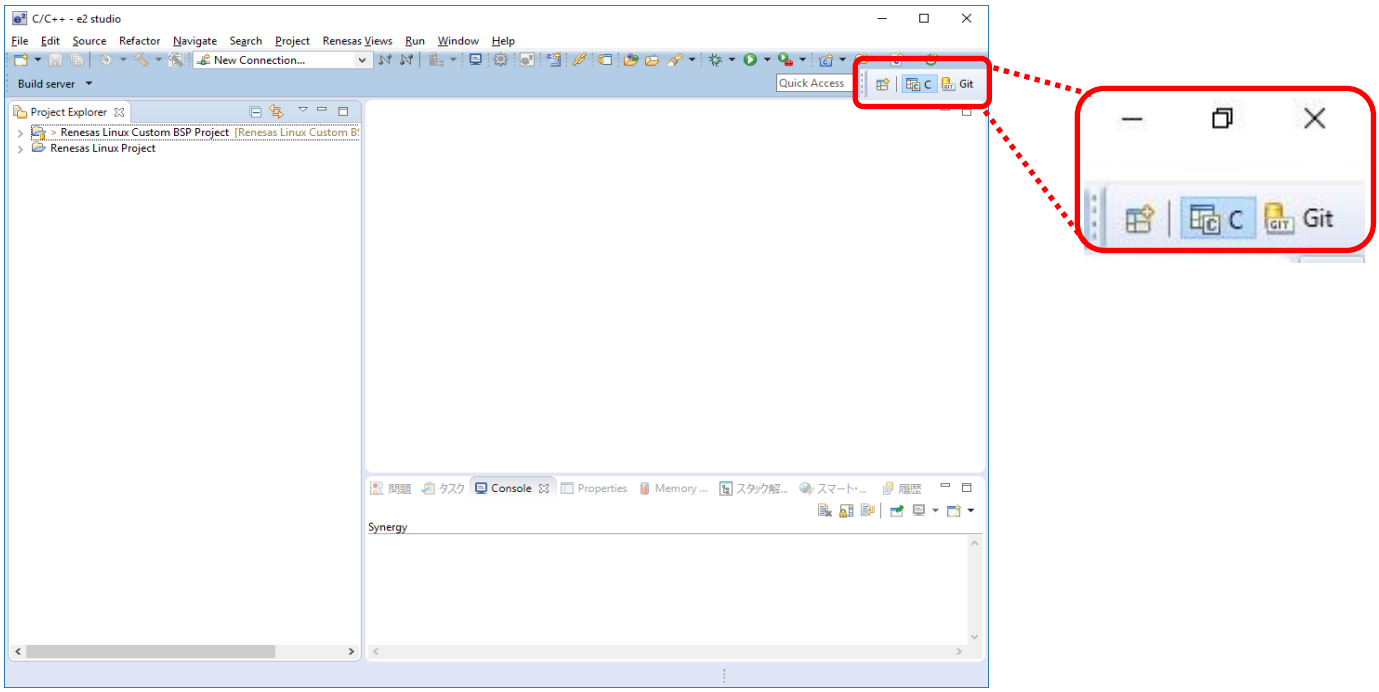




7. Select [Perspective] -> [Open Perspective] -> [Other] from [Window] on the menu bar at the top of the window to open the [Open Perspective] window. Select [Git] and click on the [OK] button.

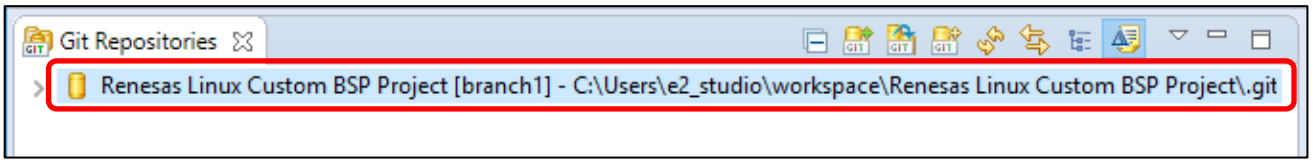


Confirm that the [Git] button is at the upper right of the window. When controlling Git after this, click on the [Git] button.

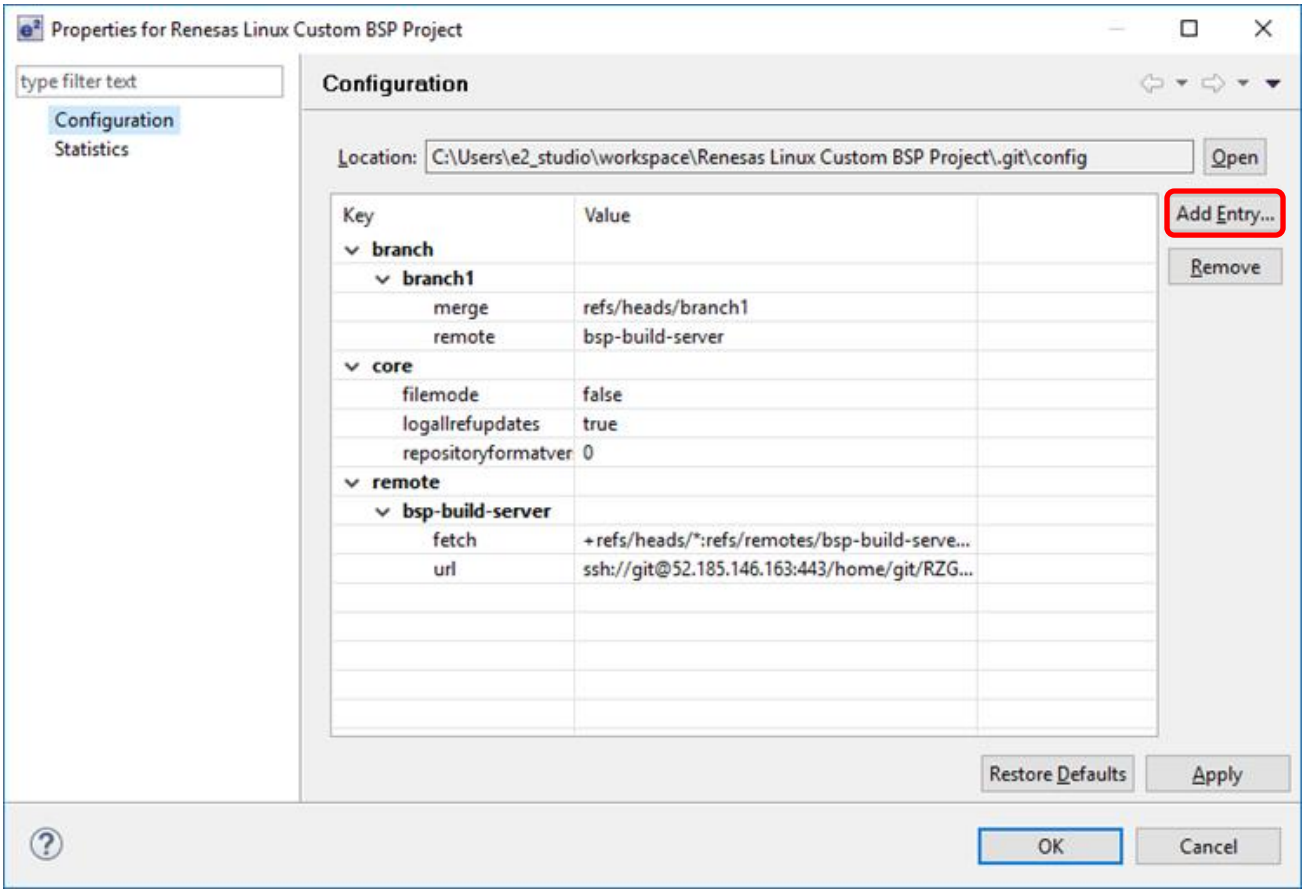


Note: To process push or pull (fetch) command, use the menu of the Linux Customization tool instead of the Git icon.

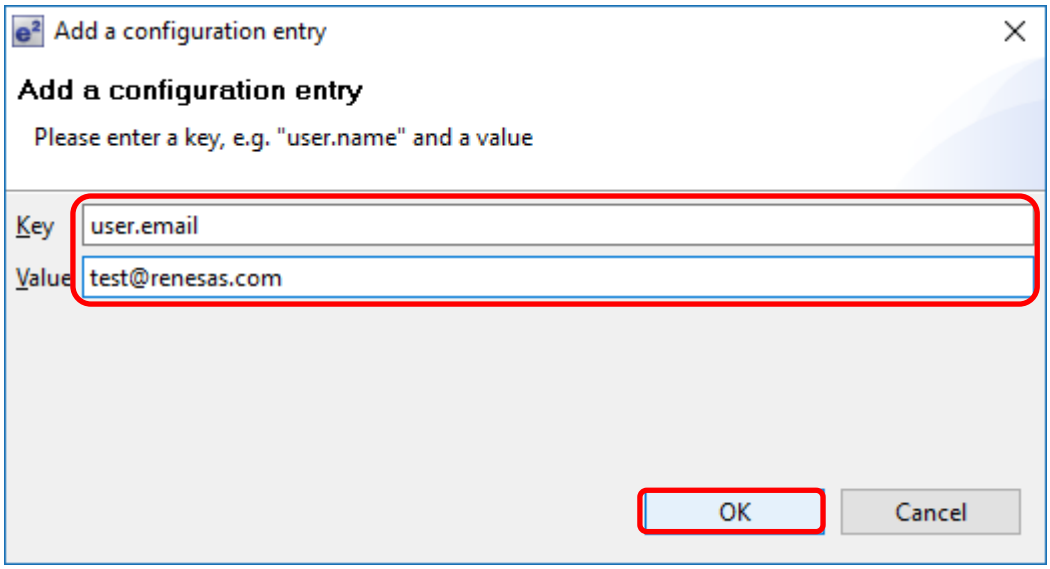
8. The [Git Repositories] tabbed page opens and the embedded Linux custom BSP project created is displayed. Left-click on the project and press [ALT + Enter] keys.



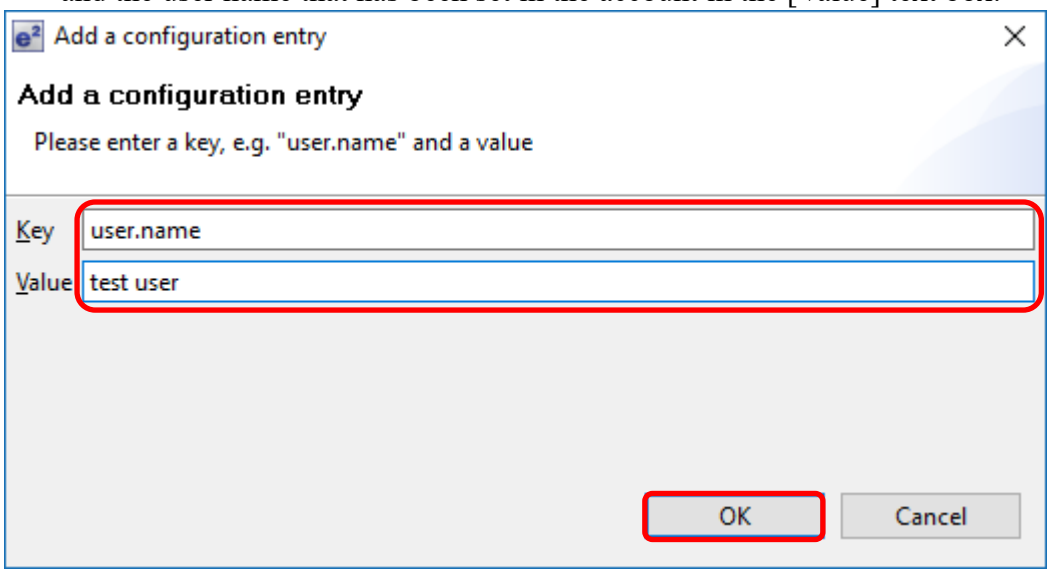
9. The [Properties for Renesas Linux Custom BPS Project] window opens. Click on the [Add Entry] button.



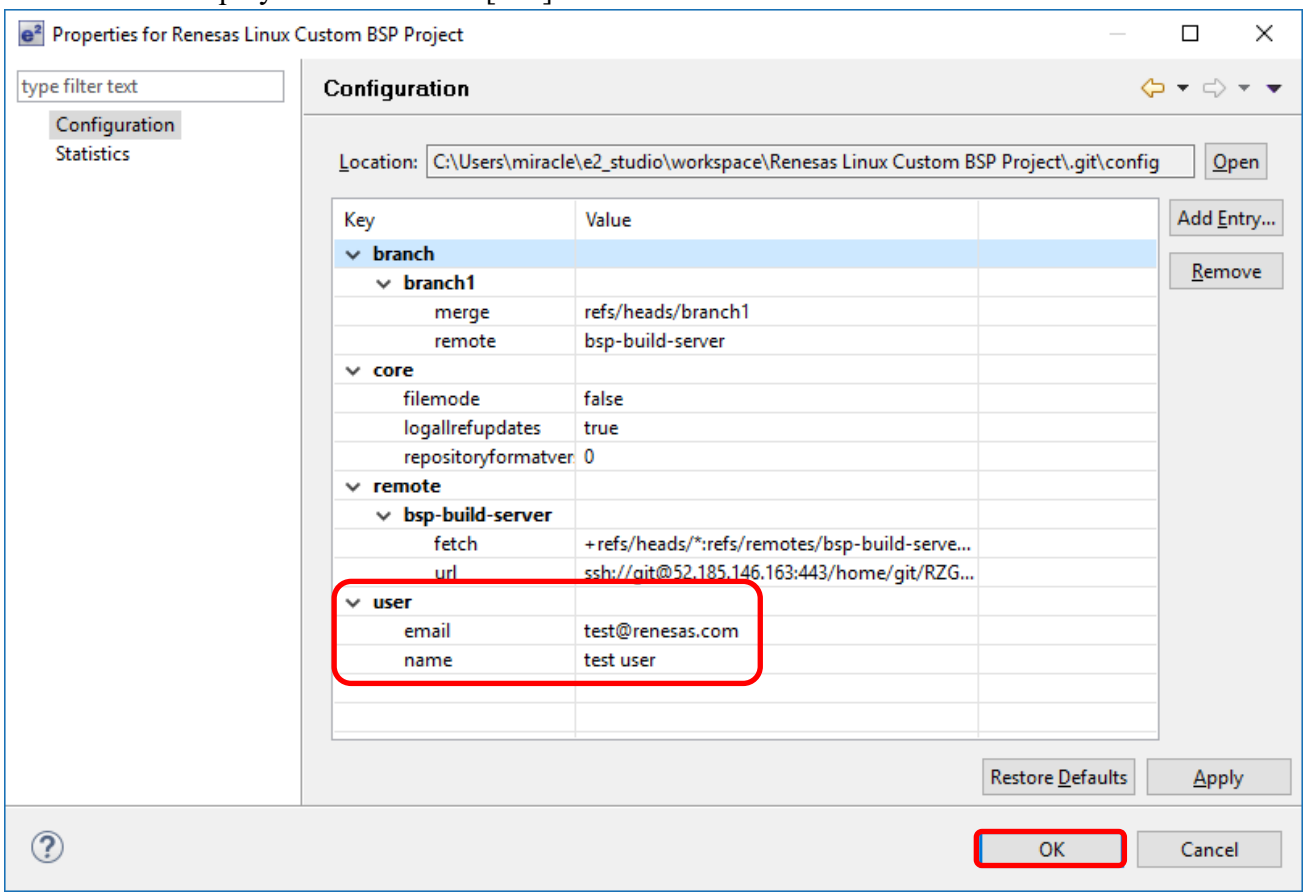
10. The [Add a configuration entry] window opens. Enter ”user.email” in the [Key] text box and the email address that has been set in the account in the [Value] text box, and click on the [OK] button.



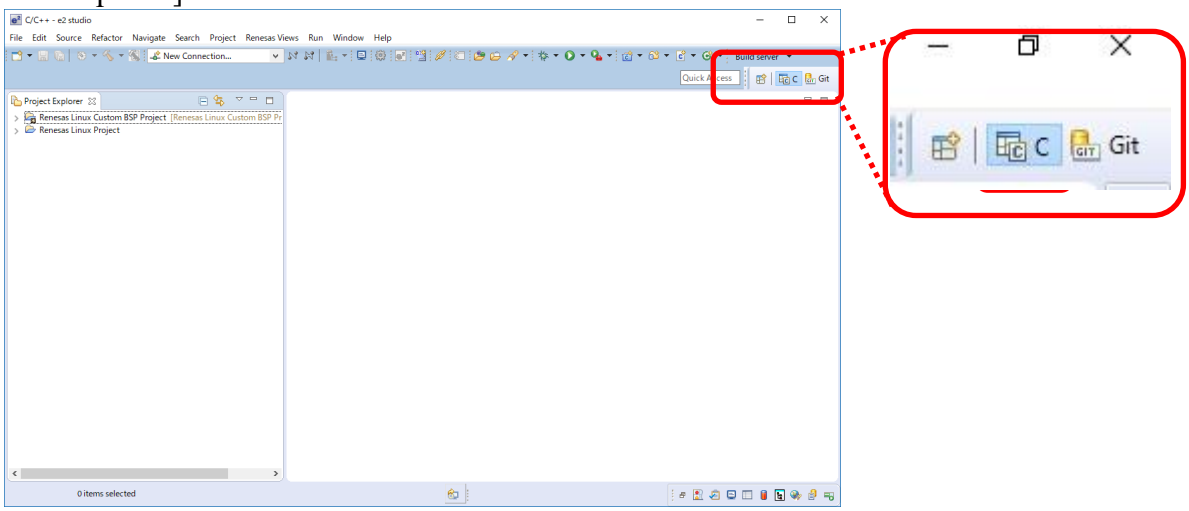
11. Open the [Add a configuration entry] window again. Enter ”user.name” in the [Key] text box and the user name that has been set in the account in the [Value] text box.



12. Confirm that the [user] section has been added to [Configuration] in the [Properties for Renesas Linux Custom BPS Project] window. Open [user] and confirm that the set email address and user name are displayed. Click on the [OK] button.



13. Click on the [C] button at the upper right of the e2 studio window to go back to the [Project Explorer] screen.

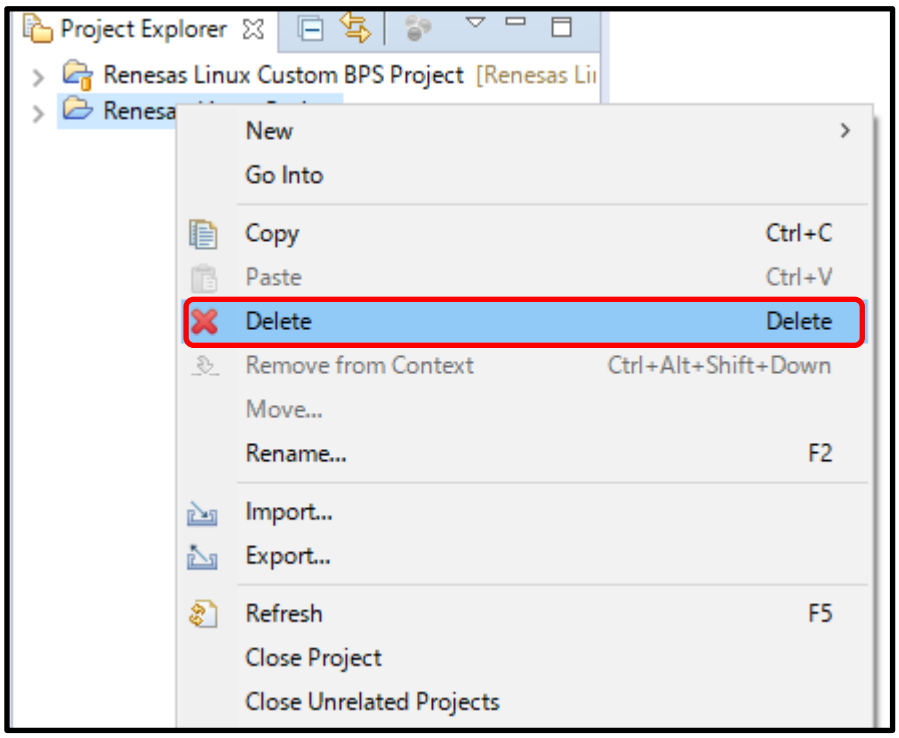


This completes creation of the embedded Linux custom BSP project.

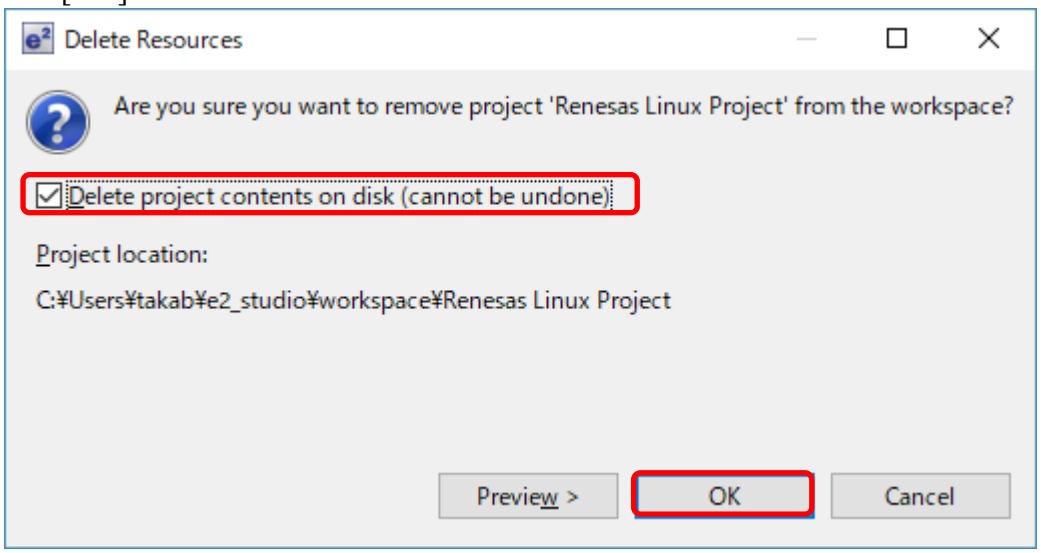
3.5.3 Deleting a Project

This subsection describes how to delete the embedded Linux project or embedded Linux custom BSP project that has been created.

1. Right-click on the project to be deleted in the [Project Explorer] pane and select [Delete] from the pop-up menu.



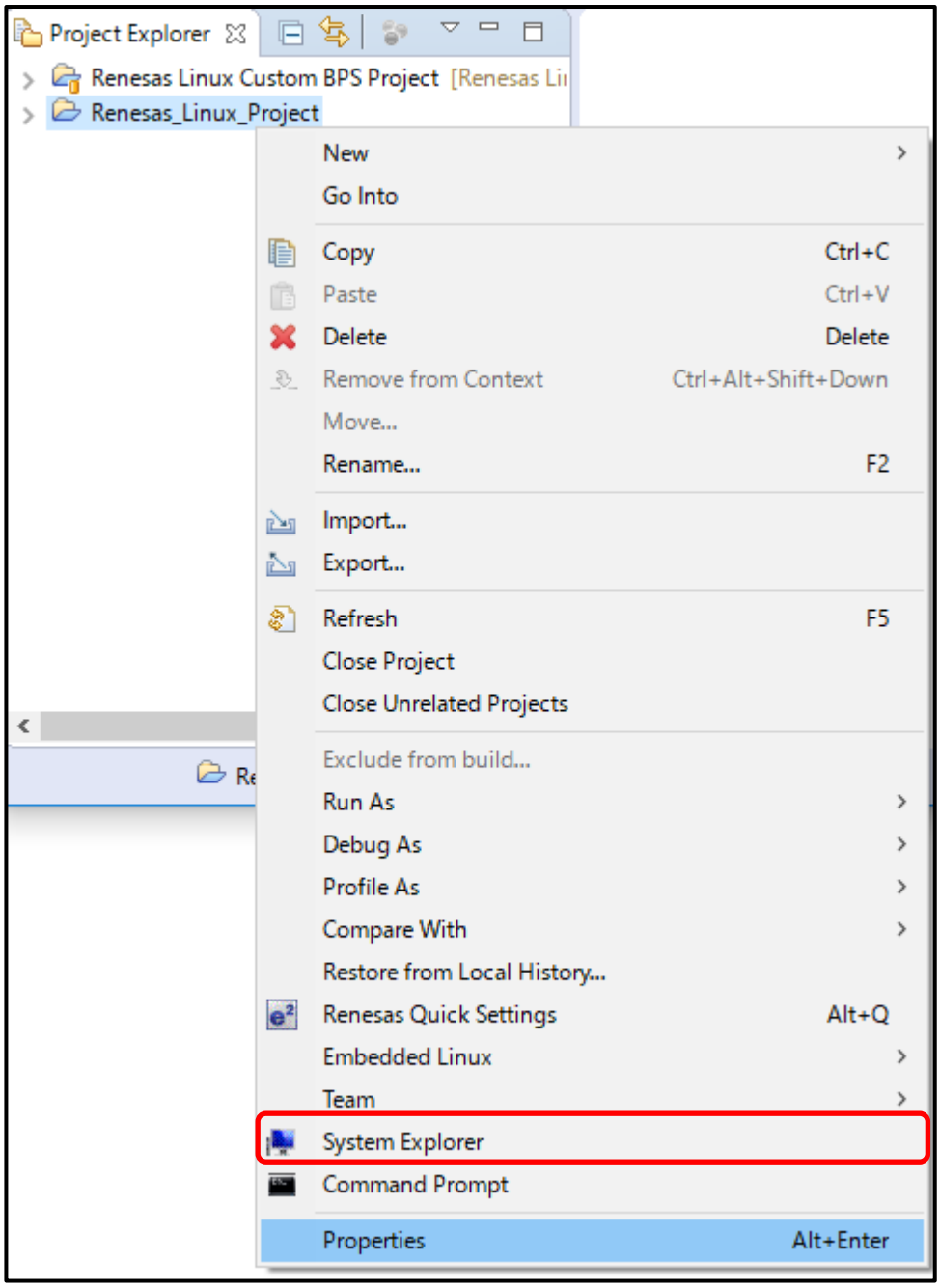
2. Click on the checkbox for [Delete project contents on disk (cannot be undone)] and click on the [OK] button.



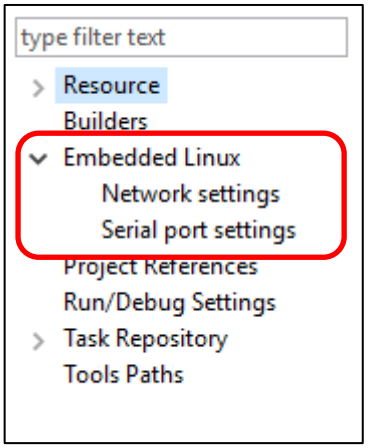
3.6 Setting Properties

This section describes the procedure for setting the properties of the project which has been created.

1. Right-click on the embedded Linux project in the [Project Explorer] pane and select [Properties] from the pop-up menu.



2. Select [Embedded Linux] to open the menu.

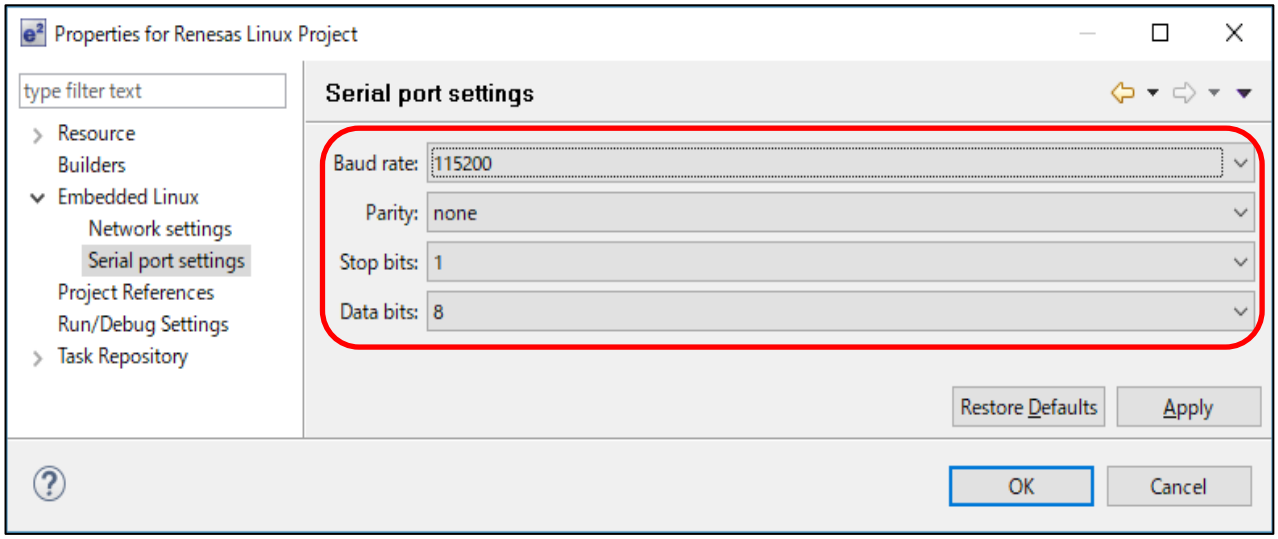


3.6.1 Setting Serial Ports

Set the serial port for communications with the target board. Set the parameters appropriately according to the target board.

Note: The following shows the parameters for the target board from iWave.

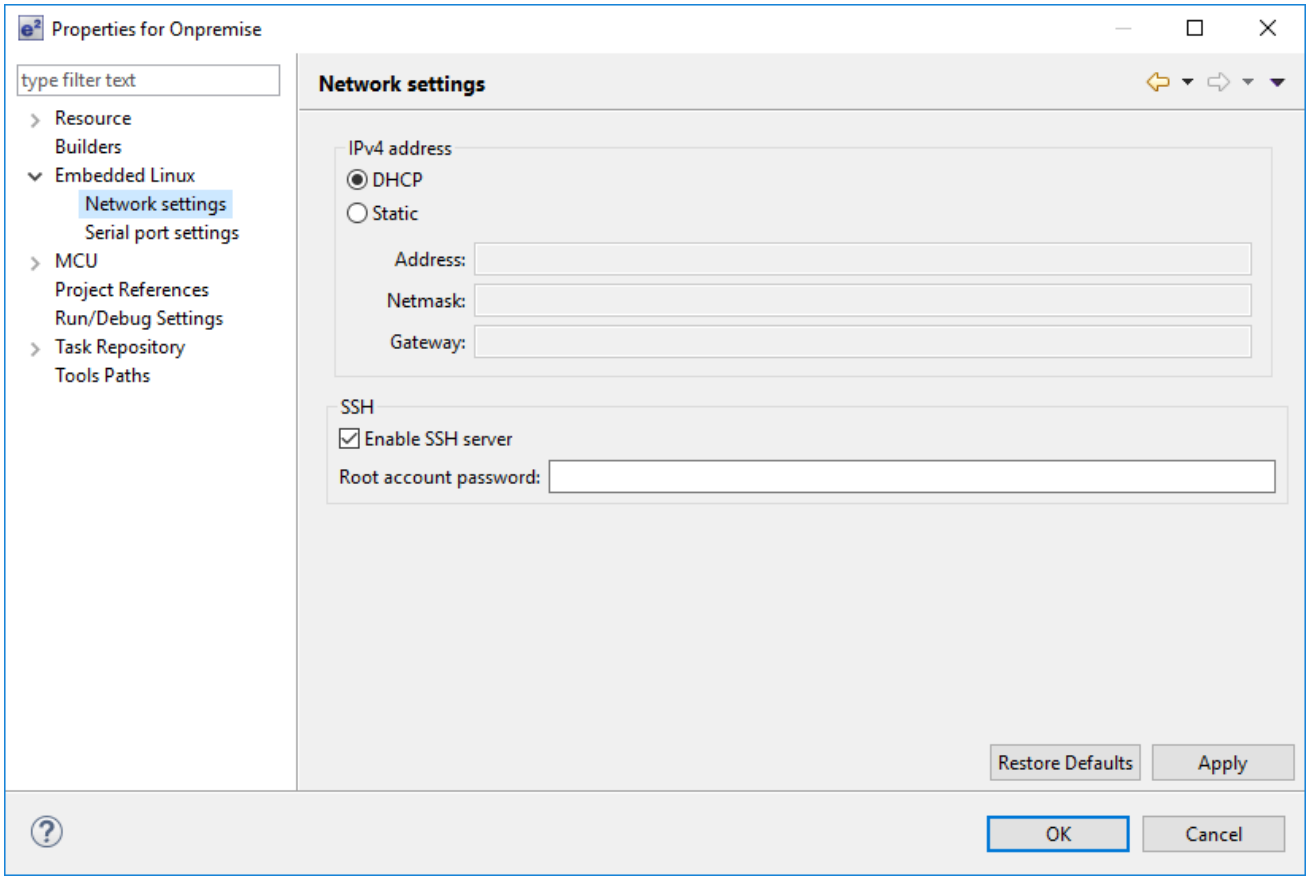
Baud rate: 115200
Parity: None
Stop bits: 1
Data bits: 8



3.6.2 Making the Network Settings

This subsection describes how to make the settings for the network on the target board when the network is used to write a user program.

Notes: 1. The settings made in this step are reflected in the image file of the results of building.
2. If you specify the password for the root account, logging in to the target board requires the password regardless of whether you are using the serial port or a network connection.
3. To disable the SSH server, uncheck [Enable SSH server]. Note that disabling the SSH server makes using the network for the writing of user programs impossible. In such a case, use the serial port to do so.



[Point for Caution on Making the Network Settings]

Enabling the SSH server allows anyone to log in to the network by simply setting the correct IP address and password for the root account (these are not set by default).

Therefore, if you wish to prevent login by a non-specific number of persons due to the field of usage of a product or for other reasons, disable the SSH server when creating the image file.

3.7 Adding

This section describes how to add programs developed by users to the embedded Linux project.
The added user programs are built by using the cross-compiler environment on the build system. For how to build the added user programs, see 3.8.1 Setting up Build and 3.8.2 Executing Build.

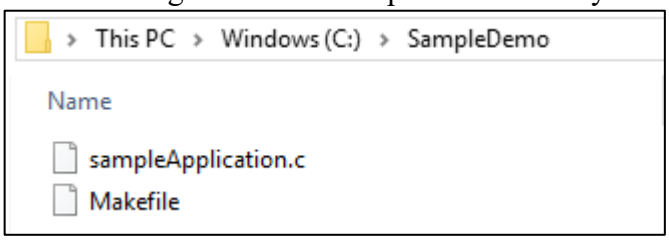
3.7.1 Adding User Programs

This subsection describes how to add programs developed by users to the embedded Linux project.
For details on how to create a user program, see the corresponding application note for the RZ/G series.

1. Create a directory on the client PC to save the following files.

File	Description
User program	Source file to be built, already built object files to be added to the root file system, or other files.
Makefile	File for holding commands, path, and environmental variables to build the added user programs

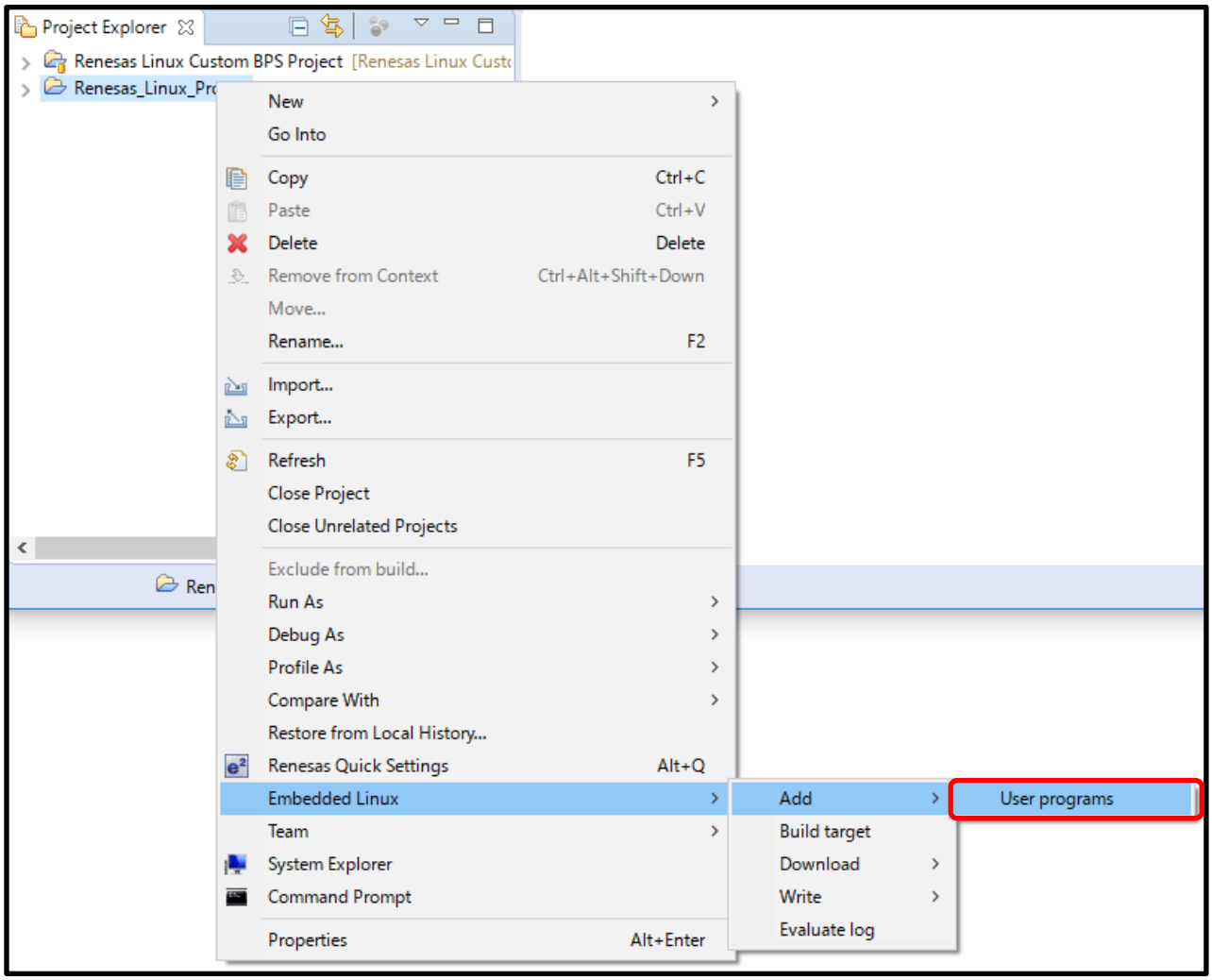
The following shows an example of a directory.



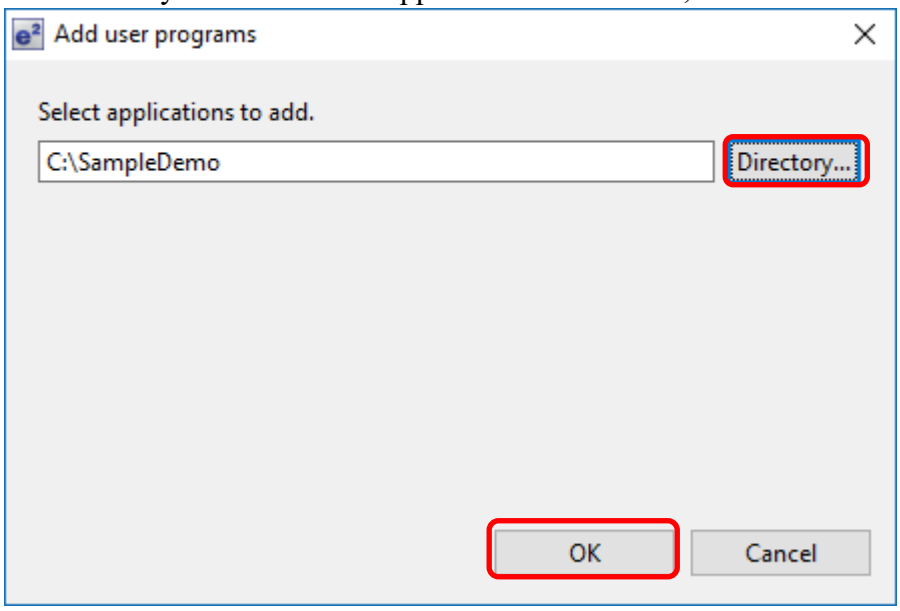
■Example of a directory

```
|--SampleDemo
    |
    |--sampleApplication.c （* a user program）
    |
    |--Makefile
```

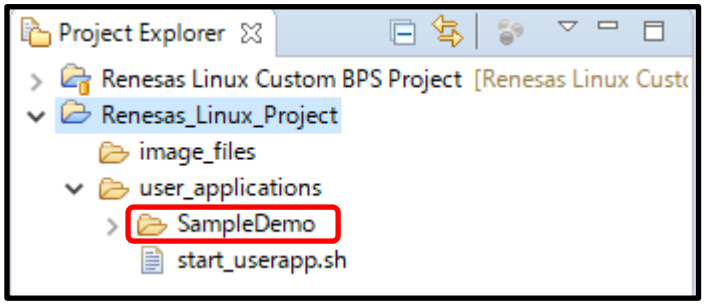
2. Right-click on the embedded Linux project in the [Project Explorer] pane and select [Embedded Linux] -> [Add] -> [User programs] from the pop-up menu.



3. When the [Add user programs] window opens, click on the [Directory] button to select a directory which holds an application to be added, and click on the [OK] button.



4. The added application can be confirmed in the [Project Explorer] pane.



3.7.2 System of Building User Programs

This subsection describes how the added user programs are built in the build server.

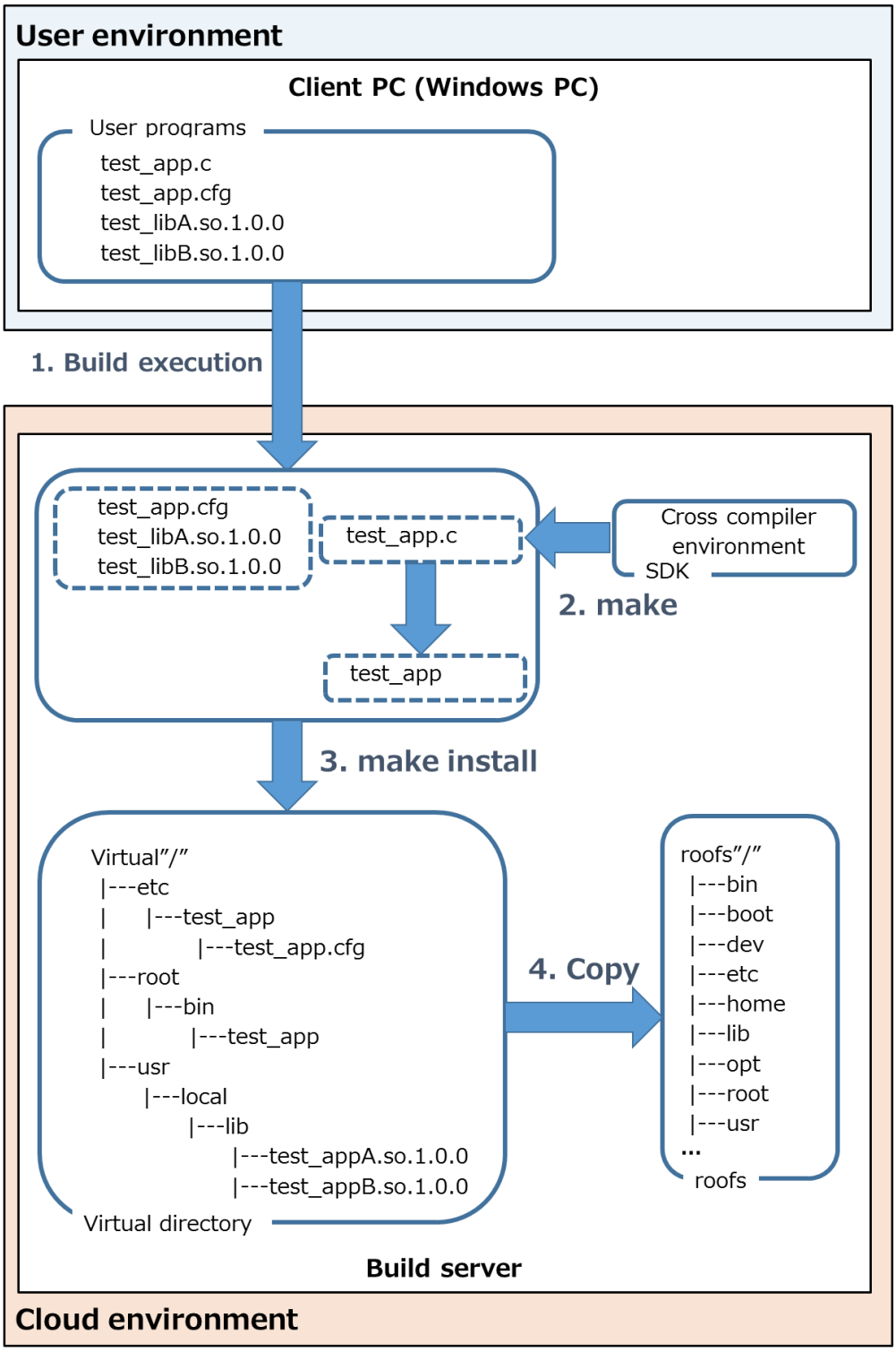


Figure 3-1 Building Sequence of User Programs

1. Build execution
   Executing build allows a set of the user programs provided on the client PC is uploaded to the build server in the build system.
   (Note: In the example shown in the above figure, a set of the user programs is uploaded.)

2. make
   When Makefile contains a description to execute build, build (make) is executed by SDK (cross-compiler environment) on the build server.
   (In the example shown in the above figure, test_app.c is built.)

3. make install
   According to the descriptions in "make install" of Makefile, the executable files, library, configuration file, and other appropriate files of the added user programs are installed in the virtual directory.
   (Note: When build is completed, the virtual directory and installed files are deleted.)

4. Copy
   If a disk image or eMMC installer has been specified as a build target during build execution in step 1 above, the files installed in the virtual directory are copied to rootfs.
   This allows the user to download the disk image or eMMC installer created by rootfs to which the desired files have been copied.
   If a user program has been specified as a build target during build execution, no files are copied to rootfs. The user can download "user program: executable binary", which is a result of compressing the virtual directory by tar.

To install the user programs to the virtual directory in the build server, specify $(DESTDIR) as the installation destination in Makefile.

Example:
install -D -m 644 test_app.cfg $(DESTDIR)/etc/test_app/test_app.cfg

For details of creating Makefile, refer to 3.7.3 Creating Makefile.

3.7.3 Creating Makefile

Makefile is a file in which the commands, path, environmental variables are set to build the user-developed programs by using the cross-compiler environment on the build system.
This subsection describes how to create Makefile.

3.7.3.1 Available Shell Commands

With Makefile, the shell commands supported by bash can be used.
(Examples: cd, makedir, cp, tar)

3.7.3.2 Available Variables

With Makefile, the variables shown in the following table can be used.

Variable	Description
CC	Loaded as gcc in the cross compiler environment.
CXX	Loaded as g++ in the cross compiler environment.
CPP	Loaded as gcc preprocessor in the cross compiler environment.
AS	Loaded as as (assembler) in the cross compiler environment.
LD	Loaded as ld (linker) in the cross compiler environment.
STRIP	Loaded as strip in the cross compiler environment.
RANLIB	Loaded as ranlib in the cross compiler environment.
OBJCOPY	Loaded as objcopy in the cross compiler environment.
OBJDUMP	Loaded as objdump in the cross compiler environment.
AR	Loaded as ar in the cross compiler environment.
NM	Loaded as nm in the cross compiler environment.
CFLAGS	Standard compiler option to be passed to gcc is expanded. An option can be added with CFLAGS +=.
CXXFLAGS	Standard compiler option to be passed to g++ is expanded. An option can be added with CXXFLAGS +=.
LDFLAGS	Standard compiler option to be passed to ld is expanded. An option can be added with LDFLAGS +=.
CPPFLAGS	Standard compiler option to be passed to gcc preprocessor is expanded. An option can be added with CPPFLAGS +=.
SDKTARGETSYSROOT	The path of the top :" /" directory where the files used by the SDK environment have been installed is expanded. Directories including lib and usr are present under this path. For example, the CC standard include path is $(SDKTARGETSYSROOT)/usr/include, the standard library path is $(SDKTARGETSYSROOT)/lib, or $(SDKTARGETSYSROOT)/usr/lib. The kernel-related files necessary for building loadable kernel modules are under $(SDKTARGETSYSROOT)/usr/src/kernel.
DESTDIR	The path of the virtual " /" directory where the user application programs are to be installed is expanded. The files under this path are copied to rootfs when the disk image is built.

3.7.3.3 Available Libraries

With Makefile, path can be set by using "package-config".
"package-config" is a method, which allows flags and paths necessary for using the library to be provided via the common interface. For the method to use package-config with Makefile, refer to 3.7.3.4 Makefile Samples.
The following table shows the libraries that can be used with package-config.

Library	Description
alsa	alsa - Advanced Linux Sound Architecture (ALSA) - Library
atk	Atk - Accessibility Toolkit
atk-bridge-2.0	atk-bridge-2.0 - ATK/D-Bus Bridge
atomic_ops	The atomic_ops project - Atomic memory update operations portable implementation
atspi-2	atspi - Accessibility Technology software library
avahi-client	avahi-client - Avahi Multicast DNS Responder (Client Support)
avahi-core	avahi-core - Avahi Multicast DNS Responder (Embeddable Stack)
avahi-glib	avahi-glib - Avahi Multicast DNS Responder (GLib Support)
avahi-gobject	avahi-gobject - Avahi Multicast DNS Responder (GLib GObject Support)
babeltrace	Babeltrace - libbabeltrace provides a reader for trace files, reading mainly the
blkid	blkid - Block device id library
bluez	BlueZ - Bluetooth protocol stack for Linux
cairo	cairo - Multi-platform 2D graphics library
cairo-egl	cairo-egl - EGL functions for cairo graphics library
cairo-fc	cairo-fc - Fontconfig font backend for cairo graphics library
cairo-ft	cairo-ft - FreeType font backend for cairo graphics library
cairo-glesv2	cairo-glesv2 - OpenGLESv2 surface backend for cairo graphics library
cairo-gobject	cairo-gobject - gobject functions for cairo graphics library
cairo-pdf	cairo-pdf - PDF surface backend for cairo graphics library
cairo-png	cairo-png - PNG functions for cairo graphics library
cairo-ps	cairo-ps - PostScript surface backend for cairo graphics library
cairo-script	cairo-script - script surface backend for cairo graphics library
cairo-svg	cairo-svg - SVG surface backend for cairo graphics library
cairo-tee	cairo-tee - tee surface backend for cairo graphics library
cairo-xcb	cairo-xcb - XCB surface backend for cairo graphics library
cairo-xcb-shm	cairo-xcb-shm - XCB/SHM functions for cairo graphics library
cairo-xlib	cairo-xlib - Xlib surface backend for cairo graphics library
cairo-xlib-xrender	cairo-xlib-xrender - Xlib Xrender surface backend for cairo graphics library
cally-1.0	Cally - Clutter Accessibility Implementation Library
check	Check - A unit test framework for C
ck-connector	ck-connector - Library for connecting new sessions to ConsoleKit
clutter-1.0	Clutter - Clutter Core Library
clutter-cogl-1.0	Clutter - Clutter Core Library
clutter-glx-1.0	Clutter - Clutter Core Library
clutter-wayland-1.0	Clutter - Clutter Core Library
clutter-x11-1.0	Clutter - Clutter Core Library
cogl-1.0	Cogl - An object oriented GL/GLES Abstraction/Utility Layer

Library	Description
cogl-2.0-experimental	Cogl - An object oriented GL/GLES Abstraction/Utility Layer
cogl-gl-1.0	Cogl - An object oriented GL/GLES Abstraction/Utility Layer
cogl-pango-1.0	Cogl - An object oriented GL/GLES Abstraction/Utility Layer
cogl-pango-2.0-experimental	Cogl - An object oriented GL/GLES Abstraction/Utility Layer
com_err	com_err - Common error description library
compositeproto	CompositeExt - Composite extension headers
connman	connman - Connection Manager
damageproto	DamageProto - Damage extension headers
dbus-1	dbus - Free desktop message bus
dbus-glib-1	dbus-glib - GLib integration for the free desktop message bus
dbus-python	dbus-python - Python bindings for D-Bus
++dfb	++DFB - C++ binding for DirectFB
direct	Direct - DirectFB base development library
directfb	DirectFB - Graphics and windowing library for the Linux frame buffer device
directfb-internal	DirectFB-Internal - Third party module support package for DirectFB.
dri2proto	DRI2Proto - DRI2 extension headers
e2p	e2p - Ext2fs userpace programs utility library
egl	egl - EGL library with Wayland Extension
expat	expat - expat XML parser
ext2fs	ext2fs - Ext2fs library
fixesproto	FixesProto - X Fixes extension headers
flac	FLAC - Free Lossless Audio Codec Library
flac++	FLAC++ - Free Lossless Audio Codec Library (C++ API)
fontconfig	Fontconfig - Font configuration and customization library
form	form - ncurses 5.9 add-on library
formw	formw - ncurses 5.9 add-on library
freetype2	FreeType 2 - A free, high-quality, and portable font engine.
fusion	Fusion IPC - High Level IPC Mechanisms
gail-3.0	Gail - GNOME Accessibility Implementation Library
gail	Gail - GNOME Accessibility Implementation Library
gbm	gbm - gbm library
gconf-2.0	gconf - GNOME Config System.
gdk-2.0	GDK - GTK+ Drawing Kit (x11 target)
gdk-3.0	GDK - GTK+ Drawing Kit
gdk-pixbuf-2.0	GdkPixbuf - Image loading and scaling
gdk-wayland-3.0	GDK - GTK+ Drawing Kit
gdk-x11-2.0	GDK - GTK+ Drawing Kit (x11 target)
gdk-x11-3.0	GDK - GTK+ Drawing Kit
gio-2.0	GIO - glib I/O library
gio-unix-2.0	GIO unix specific APIs - unix specific headers for glib I/O library
glesv2	glesv2 - OpenGL/ES 2.0 Library
gl	gl - Mesa OpenGL library
glib-2.0	GLib - C Utility Library
gmodule-2.0	GModule - Dynamic module loader for GLib
gmodule-export-2.0	GModule - Dynamic module loader for GLib
gmodule-no-export-2.0	GModule - Dynamic module loader for GLib
gnutls-extra	GnuTLS-extra - Additional add-ons for GnuTLS licensed under GPL

Library	Description
gnutls	GnuTLS - Transport Security Layer implementation for the GNU system
gobject-2.0	GObject - GLib Type, Object, Parameter and Signal Library
gpg-error	gpg-error - a library that defines common error values for all GnuPG components
gstreamer-1.0	GStreamer - Streaming media framework
gstreamer-allocators-1.0	GStreamer Allocators Library - Allocators implementation
gstreamer-app-1.0	GStreamer Application Library - Helper functions and base classes for application integration
gstreamer-audio-1.0	GStreamer Audio library - Audio helper functions and base classes
gstreamer-base-1.0	GStreamer base classes - Base classes for GStreamer elements
gstreamer-codecparsers-1.0	GStreamer codec parsers - Bitstream parsers for GStreamer elements
gstreamer-controller-1.0	GStreamer controller - Dynamic parameter control for GStreamer elements
gstreamer-egl-1.0	GStreamer EGL - GStreamer EGL library
gstreamer-fft-1.0	GStreamer FFT Library - FFT implementation
gstreamer-insertbin-1.0	GStreamer Insert Bin - Bin to automatically and insertally link elements
gstreamer-mpegts-1.0	GStreamer MPEG-TS - GStreamer MPEG-TS support
gstreamer-net-1.0	GStreamer networking library - Network-enabled GStreamer plug-ins and clocking
gstreamer-pbutils-1.0	GStreamer Base Utils Library - General utility functions
gstreamer-plugins-bad-1.0	GStreamer Bad Plugin libraries - Streaming media framework, bad plugins libraries
gstreamer-plugins-base-1.0	GStreamer Base Plugins Libraries - Streaming media framework, base plugins libraries
gstreamer-riff-1.0	GStreamer RIFF Library - RIFF helper functions
gstreamer-rtp-1.0	GStreamer RTP Library - RTP base classes and helper functions
gstreamer-rtsp-1.0	GStreamer RTSP Library - RTSP base classes and helper functions
gstreamer-sdp-1.0	GStreamer SDP Library - SDP helper functions
gstreamer-tag-1.0	GStreamer Tag Library - Tag base classes and helper functions
gstreamer-video-1.0	GStreamer Video Library - Video base classes and helper functions
gthread-2.0	GThread - Thread support for GLib
gtk+-2.0	GTK+ - GTK+ Graphical UI Library (x11 target)
gtk+-3.0	GTK+ - GTK+ Graphical UI Library
gtk+-unix-print-2.0	GTK+ - GTK+ Unix print support
gtk+-unix-print-3.0	GTK+ - GTK+ Unix print support
gtk+-wayland-3.0	GTK+ - GTK+ Graphical UI Library
gtk+-x11-2.0	GTK+ - GTK+ Graphical UI Library (x11 target)
gtk+-x11-3.0	GTK+ - GTK+ Graphical UI Library
gudev-1.0	gudev-1.0 - GObject bindings for libudev
harfbuzz	harfbuzz - HarfBuzz text shaping library
ice	ICE - X Inter Client Exchange Library
icu-i18n	icu-i18n - International Components for Unicode: Internationalization library
icu-io	icu-io - International Components for Unicode: Stream and I/O Library
icu-le	icu-le - International Components for Unicode: Layout library
icu-lx	icu-lx - International Components for Unicode: Paragraph Layout library
icu-uc	icu-uc - International Components for Unicode: Common and Data libraries
inputproto	InputProto - Input extension headers
json-c	json-c - JSON implementation in C

Library	Description
json-glib-1.0	JSON-GLib - JSON Parser for GLib
json	json-c - JSON implementation in C, compat shim. Use json-c instead.
kbproto	KBProto - KB extension headers
libcrypto	OpenSSL-libcrypto - OpenSSL cryptography library
libdaemon	libdaemon - a lightweight C library that eases the writing of UNIX daemons
libdrm	libdrm - Userspace interface to kernel DRM services
libdrm_nouveau	libdrm_nouveau - Userspace interface to nouveau kernel DRM services
libdrm_omap	libdrm_omap - Userspace interface to omap kernel DRM services
libdrm_radeon	libdrm_radeon - Userspace interface to kernel DRM services for radeon
libevdev	libevdev - Handler library for evdev events
libffi	libffi - Library supporting Foreign Function Interfaces
libgcrypt	libgcrypt - GNU crypto library
libglade-2.0	Libglade - a library for dynamically loading GLADE interface files
libinput	Libinput - Input device library
libip4tc	libip4tc - iptables IPv4 ruleset ADT and kernel interface
libip6tc	libip6tc - iptables IPv6 ruleset ADT and kernel interface
libiptc	libiptc - iptables v4/v6 ruleset ADT and kernel interface
libkmod	libkmod - Library to deal with kernel modules
libkms	libkms - Library that abstract aways the different mm interface for kernel drivers
liblzma	liblzma - General purpose data compression library
libmediactl	libmediactl - Media controller library.
libmediactl-v4l2	libmediactl - Media controller library.
libnewt	libnewt - A development library for text mode user interfaces
libnl-3.0	libnl - Convenience library for netlink sockets
libnl-cli-3.0	libnl-cli - Command Line Interface library for netlink sockets
libnl-genl-3.0	libnl-genl - Generic Netlink Library
libnl-nf-3.0	libnl-nf - Netfilter Netlink Library
libnl-route-3.0	libnl-route - Netlink Routing Family Library
liboil-0.3	liboil-0.3 - Library of Optimized Inner Loops
libopkg	libopkg - opkg package manager library
libpci	libpci - libpci
libpcrecpp	libpcrecpp - PCRECPP - C++ wrapper for PCRE
libpcre	libpcre - PCRE - Perl compatible regular expressions C library with 8 bit character support
libpcreposix	libpcreposix - PCREPosix - Posix compatible interface to libpcre
libpng16	libpng - Loads and saves PNG files
libpng	libpng - Loads and saves PNG files
libproxy-1.0	libproxy-1.0 - Proxy Configuration Library
libpulse	libpulse - PulseAudio Client Interface
libpulsemainloop-glib	libpulse-mainloop-glib - PulseAudio GLib 2.0 Main Loop Wrapper
libpulse-simple	libpulse-simple - PulseAudio Simplified Synchronous Client Interface
libsocketcan	libsocketcan - provides access to socketcan configuration interface
libsoup-2.4	libsoup - a glib-based HTTP library
libsoup-gnome-2.4	libsoup - a glib-based HTTP library
libssl	OpenSSL - Secure Sockets Layer and cryptography libraries
libtasn1	libtasn1 - Library for ASN.1 and DER manipulation

Library	Description
libtiff-4	libtiff - Tag Image File Format (TIFF) library.
libtirpc	libtirpc - Transport Independent RPC Library
libudev	libudev - Library to access udev device information
liburcu-bp	Userspace RCU BulletProof - A userspace RCU (read-copy-update) library, bulletproof version
liburcu-cds	Userspace RCU Concurrent Data Structures - Data structures leveraging RCU and atomic operations to provide efficient concurrency-aware storage
liburcu-mb	Userspace RCU Memory barriers - A userspace RCU (read-copy-update) library, memory barriers version
liburcu-qsbr	Userspace RCU QSBR - A userspace RCU (read-copy-update) library, quiescent state version
liburcu-signal	Userspace RCU signal - A userspace RCU (read-copy-update) library, signal version
liburcu	Userspace RCU - A userspace RCU (read-copy-update) library, standard version
libusb-1.0	libusb-1.0 - C API for USB device access from Linux, Mac OS X, OpenBSD, NetBSD and Windows userspace
libusb	libusb - USB access library (libusb-1.0 compat wrapper)
libv4l2subdev	libv4l2subdev - V4L2 subdevice library.
libxml-2.0	libXML - libXML library version2.
lttng-ctl	LTTng control library - The LTTng control and utility library is a library used to control the tracing sessions of a LTTng-session deamon
lttng-ust	LTTng Userspace Tracer - The LTTng Userspace Tracer (UST) is a library accompanied by a set of tools to trace userspace code.
menu	menu - ncurses 5.9 add-on library
menuw	menuw - ncurses 5.9 add-on library
meram	meram - A library for controlling SH-Mobile MERAM
mount	mount - mount library
mtdev	mtdev - Multitouch Protocol Translation Library
ncurses++	ncurses++ - ncurses 5.9 add-on library
ncurses	ncurses - ncurses 5.9 library
ncurses++w	ncurses++w - ncurses 5.9 add-on library
ncursesw	ncursesw - ncurses 5.9 library
neard	neard - NFC daemon
neon	neon - HTTP/WebDAV client library
ofono	ofono - oFono - Open Source Telephony
ogg	ogg - ogg is a library for manipulating ogg bitstreams
openssl	OpenSSL - Secure Sockets Layer and cryptography libraries and tools
orc-0.4	orc-0.4 - Library of Optimized Inner Loops Runtime Compiler
panel	panel - ncurses 5.9 add-on library
panelw	panelw - ncurses 5.9 add-on library
pangocairo	Pango Cairo - Cairo rendering support for Pango
pangoft2	Pango FT2 and Pango Fc - Freetype 2.0 and fontconfig font support for Pango
pango	Pango - Internationalized text handling
pangoxft	Pango Xft - Xft font support for Pango
pciaccess	pciaccess - Library providing generic access to the PCI bus and devices.

Library	Description
pixman-1	Pixman - The pixman library (version 1)
popt	popt - popt library.
pthread-stubs	pthread stubs - Stubs missing from libc for standard pthread functions
pygobject-2.0	PyGObject - Python bindings for GObject
python-2.7	Python - Python library
python2	Python - Python library
python	Python - Python library
Qt53DCore	Qt5 3DCore - Qt 3DCore module
Qt53DInput	Qt5 3DInput - Qt 3DInput module
Qt53DLogic	Qt5 3DLogic - Qt 3DLogic module
Qt53DQuickInput	Qt5 3DQuickInput - Qt 3DQuickInput module
Qt53DQuick	Qt5 3DQuick - Qt 3DQuick module
Qt53DQuickRender	Qt5 3DQuickRender - Qt 3DQuickRender module
Qt53DRender	Qt5 3DRender - Qt 3DRender module
Qt5Bluetooth	Qt5 Bluetooth - Qt Bluetooth module
Qt5Compositor	Qt5 Compositor - Qt Compositor module
Qt5Concurrent	Qt5 Concurrent - Qt Concurrent module
Qt5Core	Qt5 Core - Qt Core module
Qt5DBus	Qt5 DBus - Qt DBus module
Qt5Gui	Qt5 Gui - Qt Gui module
Qt5Location	Qt5 Location - Qt Location module
Qt5Multimedia	Qt5 Multimedia - Qt Multimedia module
Qt5MultimediaWidgets	Qt5 MultimediaWidgets - Qt MultimediaWidgets module
Qt5Network	Qt5 Network - Qt Network module
Qt5Nfc	Qt5 Nfc - Qt Nfc module
Qt5OpenGLExtensions	Qt5 OpenGLExtensions - Qt OpenGLExtensions module
Qt5OpenGL	Qt5 OpenGL - Qt OpenGL module
Qt5Positioning	Qt5 Positioning - Qt Positioning module
Qt5PrintSupport	Qt5 PrintSupport - Qt PrintSupport module
Qt5Qml	Qt5 Qml - Qt Qml module
Qt5Quick	Qt5 Quick - Qt Quick module
Qt5QuickTest	Qt5 QuickTest - Qt QuickTest module
Qt5QuickWidgets	Qt5 QuickWidgets - Qt QuickWidgets module
Qt5Script	Qt5 Script - Qt Script module
Qt5ScriptTools	Qt5 ScriptTools - Qt ScriptTools module
Qt5Sensors	Qt5 Sensors - Qt Sensors module
Qt5SerialPort	Qt5 SerialPort - Qt SerialPort module
Qt5Sql	Qt5 Sql - Qt Sql module
Qt5Svg	Qt5 Svg - Qt Svg module
Qt5Test	Qt5 Test - Qt Unit Testing Library
Qt5WaylandClient	Qt5 WaylandClient - Qt WaylandClient module
Qt5WebChannel	Qt5 WebChannel - Qt WebChannel module
Qt5WebSockets	Qt5 WebSockets - Qt WebSockets module
Qt5Widgets	Qt5 Widgets - Qt Widgets module
Qt5XmlPatterns	Qt5 XmlPatterns - Qt XmlPatterns module
Qt5Xml	Qt5 Xml - Qt Xml module
quota	quota - Quota management library

Library	Description
randrproto	RandrProto - Randr extension headers
recordproto	RecordProto - Record extension headers
renderproto	RenderProto - Render extension headers
samplerate	samplerate - An audio Sample Rate Conversion library
sbc	SBC - SBC library
shvio	shvio - A library for controlling SH-Mobile VIO/VEU
slang	slang - S-Lang programming library and interpreter
sm	SM - X Session Management Library
sndfile	sndfile - A library for reading and writing audio files
speexdsp	speexdsp - Speexdsp is a speech processing library that goes along with the Speex codec
speex	speex - Speex is an audio codec tuned for speech
sqlite3	SQLite - SQL database engine
sqlite	SQLite - SQL database engine
ss	ss - Subsystem command parsing library
taglib_c	TagLib C Bindings - Audio meta-data library (C bindings)
taglib	TagLib - Audio meta-data library
tcl	Tool Command Language - Tcl is a powerful, easy-to-learn dynamic programming language, suitable for a wide range of uses.
theoradec	theora - Theora video codec (decoder)
theoraenc	theora - Theora video codec (encoder)
theora	theora - Theora video codec
tic	tic - ncurses 5.9 add-on library
ticw	ticw - ncurses 5.9 add-on library
tinfo	tinfo - ncurses 5.9 terminal interface library
tslib-1.0	tslib - Touchscreen Access Library
tslib	tslib - Touchscreen Access Library
uiomux	uiomux - A library for sharing access to UIO devices
uuid	uuid - Universally unique id library
videoproto	VideoProto - Video extension headers
vorbisenc	vorbisenc - vorbisenc is a library that provides a convenient API for setting up an encoding environment using libvorbis
vorbisfile	vorbisfile - vorbisfile is a library that provides a convenient high-level API for decoding and basic manipulation of all Vorbis I audio streams
vorbisidec	vorbisidec - vorbisidec is the integer Ogg Vorbis library
vorbis	vorbis - vorbis is the primary Ogg Vorbis library
wayland-client	Wayland Client - Wayland client side library
wayland-cursor	Wayland Cursor - Wayland cursor helper library
wayland-egl	wayland-egl - Mesa wayland-egl library
wayland-kms	wayland-kms - wayland-kms library
wayland-scanner	Wayland Scanner - Wayland scanner
wayland-server	Wayland Server - Server side implementation of the Wayland protocol
weston	Weston Plugin API - Header files for Weston plugin development
x11	X11 - X Library
x11-xcb	X11 XCB - X Library XCB interface
xau	Xau - X authorization file management libary
xcb-atom	XCB Atom library - XCB atom cache

Library	Description
xcb-aux	XCB Aux library - XCB convenient functions
xcb-composite	XCB Composite - XCB Composite Extension
xcb-damage	XCB Damage - XCB Damage Extension
xcb-dpms	XCB DPMS - XCB DPMS Extension
xcb-dri2	XCB DRI2 - XCB DRI2 Extension
xcb-dri3	XCB DRI3 - XCB DRI3 Extension
xcb-event	XCB Event library - XCB event callback interface
xcb-ewmh	XCB EWMH library - XCB EWMH binding
xcb-glx	XCB GLX - XCB GLX Extension
xcb-icccm	XCB ICCCM library - XCB ICCCM binding
xcb-image	XCB Image library - XCB image convenience library
xcb-keysyms	XCB Keysyms library - XCB Keysyms
xcb-present	XCB Present - XCB Present Extension
xcb-proto	XCB Proto - X protocol descriptions for XCB
xcb-randr	XCB RandR - XCB RandR Extension
xcb-record	XCB Record - XCB Record Extension
xcb-renderutil	XCB RenderUtil library - XCB RENDER-extension utilities library
xcb-render	XCB Render - XCB Render Extension
xcb-res	XCB Res - XCB X-Resource Extension
xcb-screensaver	XCB Screensaver - XCB Screensaver Extension
xcb-shape	XCB Shape - XCB Shape Extension
xcb-shm	XCB Shm - XCB Shm Extension
xcb-sync	XCB Sync - XCB Sync Extension
xcb-util	XCB Util Core library - XCB util core interface
xcb	XCB - X-protocol C Binding
xcb-xevie	XCB Xevie - XCB Xevie Extension
xcb-xf86dri	XCB XFree86-DRI - XCB XFree86-DRI Extension
xcb-xfixes	XCB XFixes - XCB XFixes Extension
xcb-xinerama	XCB Xinerama - XCB Xinerama Extension
xcb-xkb	XCB XKB - XCB Keyboard Extension (EXPERIMENTAL)
xcb-xprint	XCB Xprint - XCB Xprint Extension
xcb-xtest	XCB XTEST - XCB XTEST Extension
xcb-xvmc	XCB XvMC - XCB XvMC Extension
xcb-xv	XCB Xv - XCB Xv Extension
xcomposite	Xcomposite - X Composite Extension Library
xcursor	Xcursor - X Cursor Library
xdamage	Xdamage - X Damage Library
xdmcp	Xdmcp - X Display Manager Control Protocol library
xextproto	XExtProto - XExt extension headers
xext	Xext - Misc X Extension Library
xfixes	Xfixes - X Fixes Library
xft	Xft - X FreeType library
xineramaproto	XineramaProto - Xinerama extension headers
xinerama	Xinerama - The Xinerama Library
xi	Xi - X Input Extension Library
xkbcommon-x11	xkbcommon-x11 - XKB API common to servers and clients - X11 support
xkbcommon	xkbcommon - XKB API common to servers and clients

Library	Description
xproto	Xproto - Xproto headers
xrandr	Xrandr - X RandR Library
xrender	Xrender - X Render Library
xtables	xtables - Shared Xtables code for extensions and iproute2
xtst	Xtst - The Xtst Library
xv	Xv - The Xv Library
zlib	zlib - zlib compression library

3.7.3.4 Makefile Samples

This subsection shows the example of Makefile for each of the following cases and describes the point of creating Makefile.

1. Build loadable kernel modules.
2. Build using package-config.
3. Specify the building order of multiple user programs.
4. Refer to the library and header file of the package added with Yocto recipe.
5. Build a proven program built on another Linux environment.
6. Store the file not to be built in rootfs.

For how to add user programs, refer to 3.7.1 Adding User Programs.

1. Build loadable kernel modules.
   This is an example of Makefile to performs the following operation.

Operation:
Build the source file of Kernel and store the built object (.ko file) as loadable kernel module in specify folder of rootfs.

```
#####################################################
# makefile
#####################################################
KERNELRELEASE ?= 4.4.55-cip3
KERNELDIR        ?= $(SDKTARGETSYSROOT)/usr/src/kernel

include $(KERNELDIR)/.config

TARGET = hello-procfs.ko

obj-m += hello-procfs.o

all:
make -C $(KERNELDIR) M=$(PWD)

clean:
make -C $(KERNELDIR) M=$(PWD) clean

install:
mkdir -p $(DESTDIR)/lib/modules/$(KERNELRELEASE)/extra
install -t $(DESTDIR)/lib/modules/$(KERNELRELEASE)/extra $(TARGET)
```

Set the directory Kernel is stored.

Execute make with C option (move to KERNELDIR before make).

Create the extra folder under "$(DESTDIR)/lib/modules/$(KERNELRELEASE)" and store the loadable kernel module(hello-procfs.ko) there.
"KERNELRELEASE" is "4.4.55-cip3" in this sample. It is necessary to change by BSP version as below.
ver1.x.x : 3.10.31-ltsi
ver2.x.x : 4.4.55-cip3

The directory structure in this sample is below.
hello-proc-module (*This is the directory to be added to the embedded linux project
├ Makefile
└ hello-procfs.c

2. Build using package-config.
   This is an example of Makefile to build the source file of application using package-config.

```
#####################################################
# makefile
#####################################################
SOURCE=hello-gtk.c

all: hello-gtk

hello-gtk: $(SOURCE)
$(CC) `pkg-config --cflags gtk+-3.0` `pkg-config --libs gtk+-3.0` -o $@ $<

clean:
rm -f hello-gtk

install:
install -D -m 755 hello-gtk $(DESTDIR)/home/root/bin/hello-gtk
```

Specify "gtk+-3.0" to package-config to get options for the library.

Specify to store the building result executable file "hello-gtk" in /home/root/bin/hello-gtk folder under rootfs.

The directory structure in this sample is below.

```
hello-proc-module (*This is the directory to be added to the embedded linux project
  ├ Makefile
  └ hello-gtk.c
```

By specifying "gtk + -3.0" for pakage-config and getting options, the options is not needed to be described in Makefile. It allows to reduce the environment dependency of Makefile.

3. Specify the building order of multiple user programs.
   This is an example of Makefile to specify the building order of multiple user programs.

```
#####################################################
# makefile
#####################################################
SUBDIRS=hello-gtk hello-proc-module

all:
for d in $(SUBDIRS); do\
(cd $$d; make);  \
done

install:
for d in $(SUBDIRS); do\
(cd $$d; make install DESTDIR=$(DESTDIR));\
done

clean:
for d in $(SUBDIRS); do\
(cd $$d; make clean);  \
done
```

Specify the building order of the subdirectories.

The directory structure in this sample is below.

```
user_apps (*This is the directory to be added to the embedded linux project
├ Makefile
├ hello-gtk
└ hello-proc-module
```

For explicitly specifying the build order of the user programs, create a top directory (user_apps) and subdirectories, store each user program (hello-gtk, hello-proc-module) in the corresponding subdirectory. Then describe in Makefile, placed in top directory, the order of calling Makefiles in each subdirectory.

In this sample, build is executed in order of hello-gtk->hello-proc-module depending on setting of SUBDIRS.

4. Refer to the library and header file of the package added with Yocto recipe.
This is an example of Makefile to refer to the library and header file of the package added with Yocto recipe in user program.
In this sample, the package added by Yocto recipe is register to sysroot as name of "mygtk".

```
#####################################################
# makefile
#####################################################
SYSROOTDIR ?= $(HOME)/RZG1E/iWave/1.2.1/bsp/build/tmp/sysroots/iwg22m
SOURCE=sample-mygtk.c

all: sample-mygtk

sample-mygtk: $(SOURCE)
$(CC) --sysroot=$(SYSROOTDIR) -I$(SYSROOTDIR)/usr/include/mygtk -
L$(SYSROOTDIR)/usr/lib -lmygtk -lgtk-3 -o $@ $<

clean:
rm -f sample-mygtk
```

Specify the directory of sysroot as the full path.

The directory structure in this sample is below.

```
sample_mygtk (*This is the directory to be added to the embedded linux project
  ├ Makefile
  └ sample-mygtk.c
```

The library and header file of the package added with Yocto recipe are stored in the relative path of the sysroot directory shown below.
$(HOME)/<chip name> /<board name>/<version>/bsp/build/tmp/sysroots/<MACHINE Configuration>

Therefore, for refer to the library and header file of the package added with Yocto recipe, set the relative path of the sysroot directory to "SYSROOTDIR" and add "$(SYSROOTDIR)" to the head of the code that refer to the library and header file.

By including the package added with Yocto recipe in the SDK, it becomes possible to specify by package-config. An example of a Makefile in that case is shown below.
For how to include in SDK, refer to section 3.16.

```
#######################################################
# makefile
#######################################################
SOURCE=sample-mygtk.c

all: sample-mygtk

sample-mygtk: $(SOURCE)
$(CC) `pkg-config --cflags mygtk` `pkg-config --libs mygtk` -o $@ $<

clean:
rm -f sample-mygtk

install:
install -D -m 755 sample-mygtk $(DESTDIR)/home/root/bin/sample-mygtk
```

Specify to "mygtk" by package-config.

5. Build a proven program built on another Linux environment.

This is an example of Makefile to build in this service a proven program built on another Linux environment.

```
#######################################################
# makefile
#######################################################
all:
make -f makefile_org

clean:
make -C $(KERNELDIR) M=$(PWD) clean

install:
```

Execute make to makefile named "makefile_org".

The directory structure in this sample is below.

```
test_build (*This is the directory to be added to the embedded linux project)
  ├ Makefile
  ├ makefile_org
  └ test_build.c
```

The proven programs built on another Linux environment are "test_build.c" and "makefile_org". The "makefile_org" is makefile for build on another Linux environment. Change the name of makefile like "makefile_org" as shown above since the file name of Makefile (or makefile) is used in this service. Then realize build by combined with Makefile shown above.

6. Store the file not to be built in rootfs

This is an example of Makefile to only store the file (such as * .jpg) that is not to be built into rootfs without building it.

```
#######################################################
# makefile
#######################################################
INSTALLDIR=/home/root/bin/

all:
echo "Stored JPEG file."

install:
install -d $(DESTDIR)$(INSTALLDIR)
cp -r test.jpg $(DESTDIR)$(INSTALLDIR)
```

Specify file path.

Output a message

The directory structure in this sample is below.

```
stored_files (*This is the directory to be added to the embedded linux project)
  ├ Makefile
  └ test.jpg
```

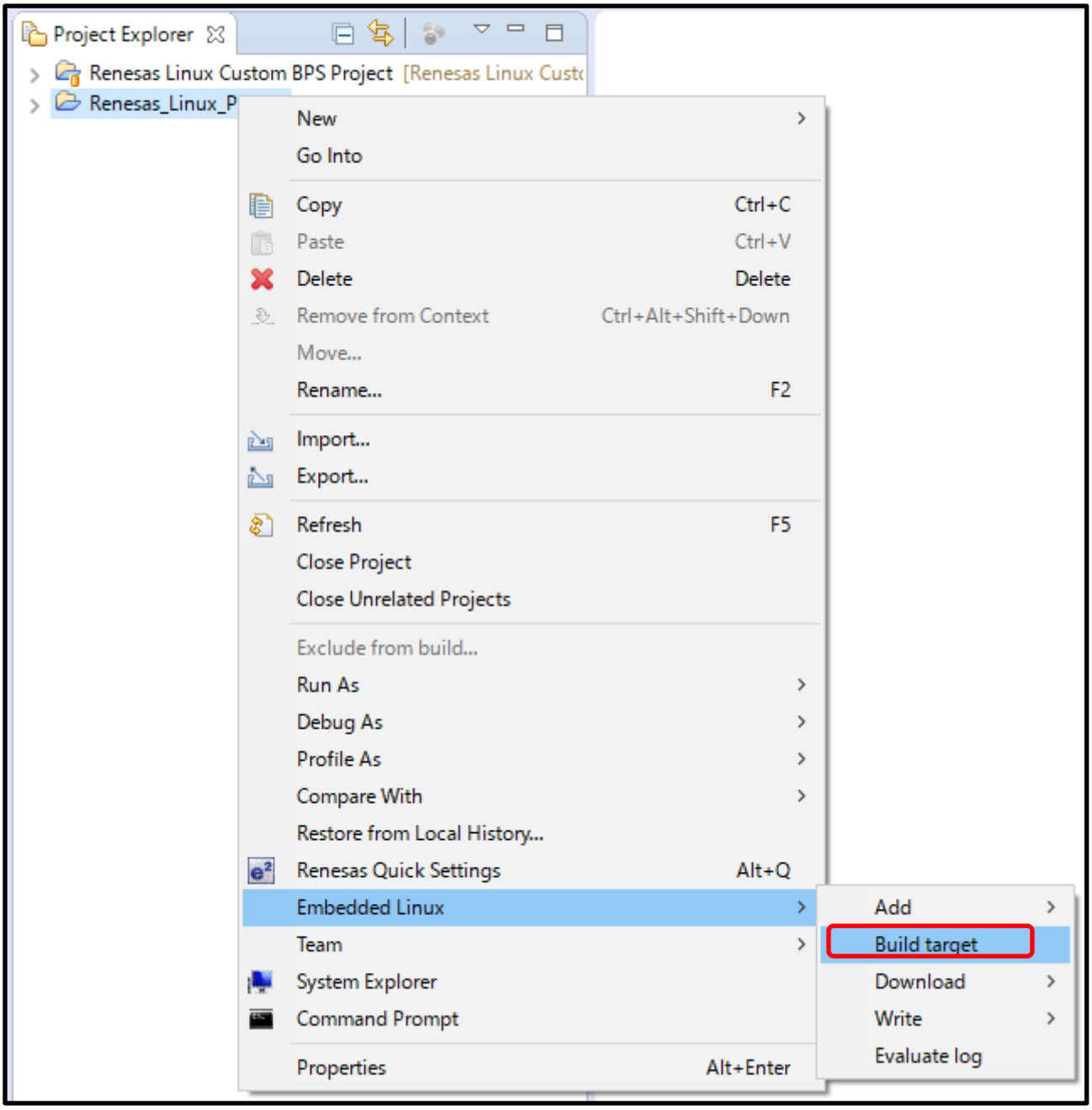
The test.jpg is stored to folder in rootfs specified in INSTALLDIR.

3.8 Building

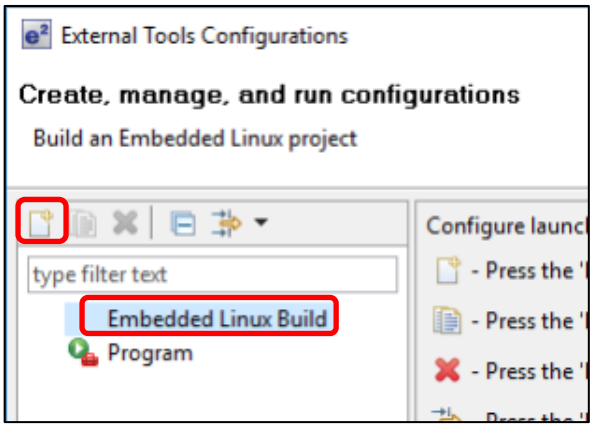
This section describes the procedure for building the user program that has been added so that it runs with the selected BSP by using the build system in the Build server.

3.8.1 Setting up Build

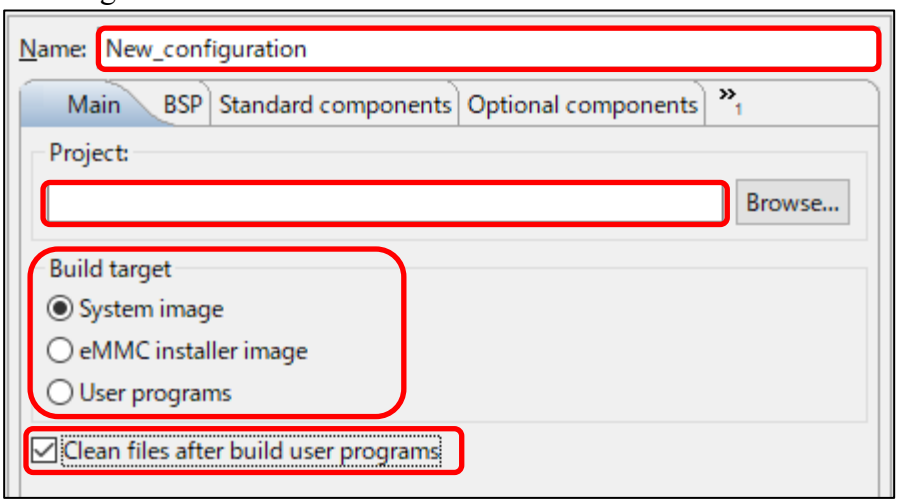
1. Right-click on the embedded Linux project in the [Project Explorer] pane and select [Embedded Linux] -> [Build target] from the pop-up menu.



2. The [External Tools Configurations] window appears. Select [Embedded Linux Build] and click on the [New launch configuration] icon at the upper left.



3. The page for setting build appears. Enter the desired name.
   Then click on the [Main] tab and specify the embedded Linux project to be built and the build target.



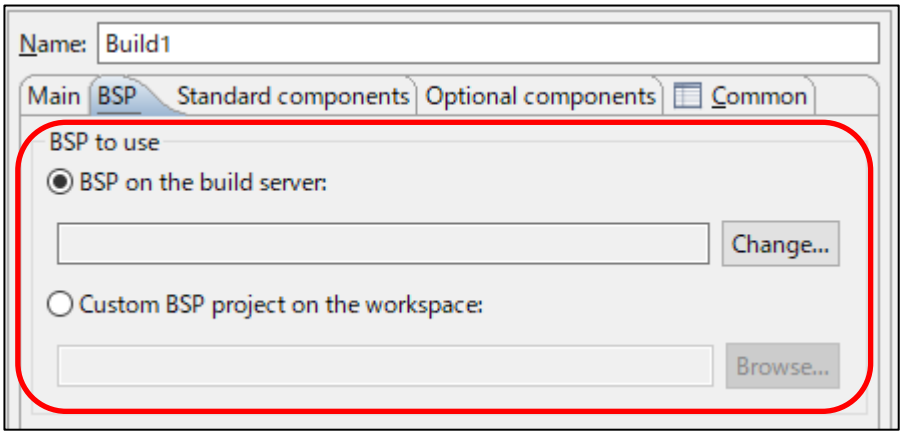
If you wish to delete the source file or files created in the midst of the build process on the build server after building the user program, check [Clean files after build user programs].
With this setting, full building of user programs always proceeds. If you wish to proceed with differential building, uncheck [Clean files after build user programs].

Build Target	Description
System image	Builds an image file to be written to an SD card. For details on how to use disk images which have been built, see 3.10.1 Writing to an SD Card.
eMMC installer image	Builds an image file to be installed in an eMMC on the target board. For details on how to use image files which have been built, see 3.10.2 Writing to an eMMC.
User programs	Builds the user program as a single body. For details on how to use files which have been built in this way, see 3.10.3 Writing a User Program.

[Point for Caution on Setting up Build]

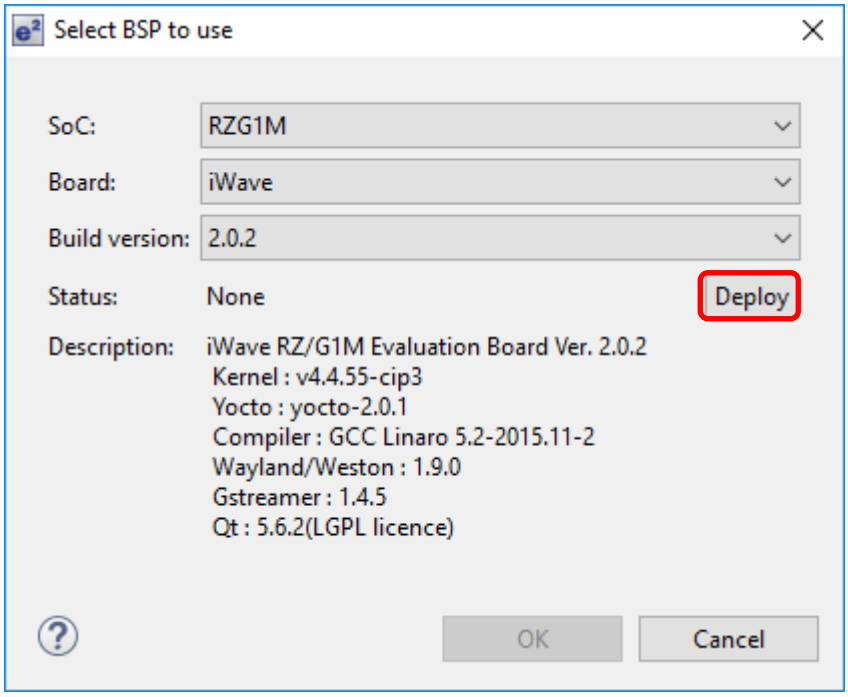
Be sure to select the project by clicking on the [Browse] button.

4. Next, click on the [BSP] tab to select the BSP used for build.

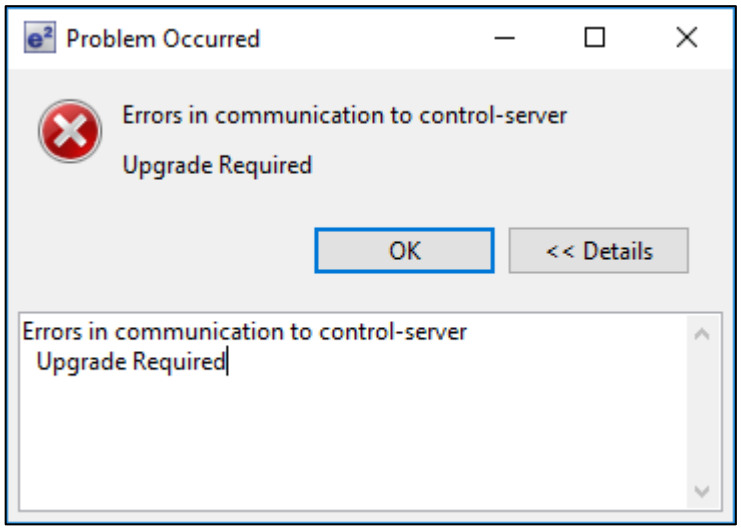


Parameter	Description
BSP on the build server	Uses the BSP on the build server for build. Click on the [Change] button to open the page for selecting the BSP. Select the chip, board, and build version appropriate for the BSP used for build.
Custom BSP project on the workspace	Uses the BSP that has been customized by the embedded Linux custom BSP project for build. Click on the [Browse] button and select the embedded Linux custom BSP project that has customized the BSP.

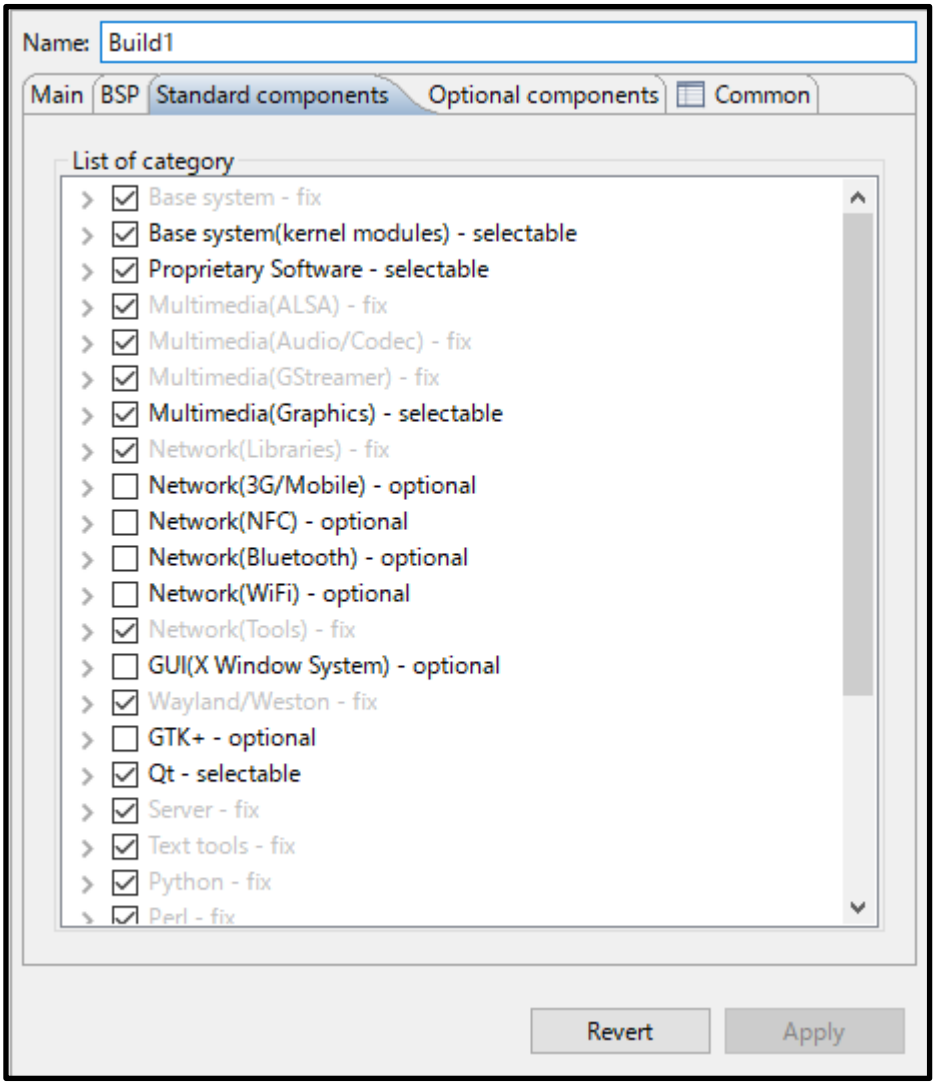
If the [Status] is [None] on the [Select BSP to use] window, click on the [Deploy] button.
And wait for [Status] to become [Available].



An attempt to select [BSP on the build server:] may lead to display of the following error message. This error occurs because the version of the RZ/G Linux platform tools is not the latest. In such cases, refer to section 2.1.5.3 Updating the RZ/G Linux Platform Tool Plug-ins, and proceed with updating the version.



5. Click on the [Standard components] tab and select the components to be built. The components selected here will be built and incorporated in the image file.



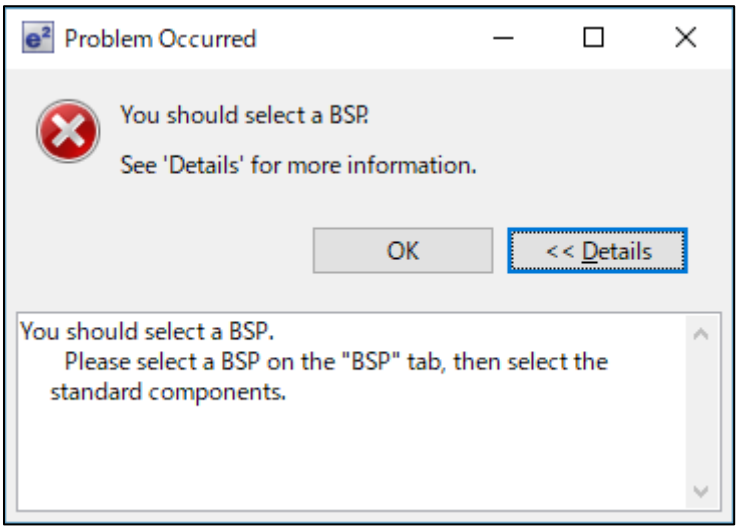
[Note 1 on Selecting Standard Components]

[Qt] of [Qt – selectable] in the list of standard components refers to the Qt under the license of LGPL.

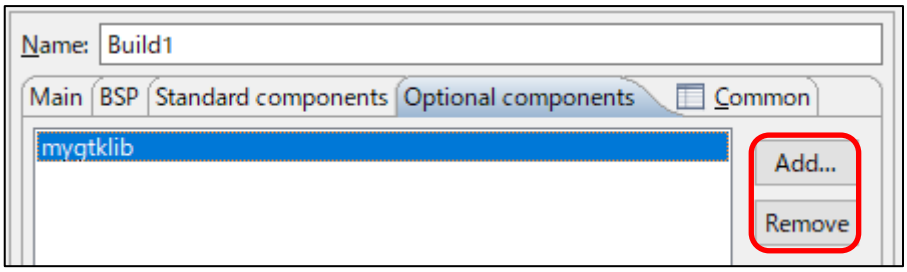
[Note 2 on Selecting Standard Components]

When writing to a target board as in writing to an eMMC or writing a user program, be sure to select [Development Tools].

An attempt to click on the [Standard components] tab without making a selection on the [BSP] tabbed page leads to display of the following warning.



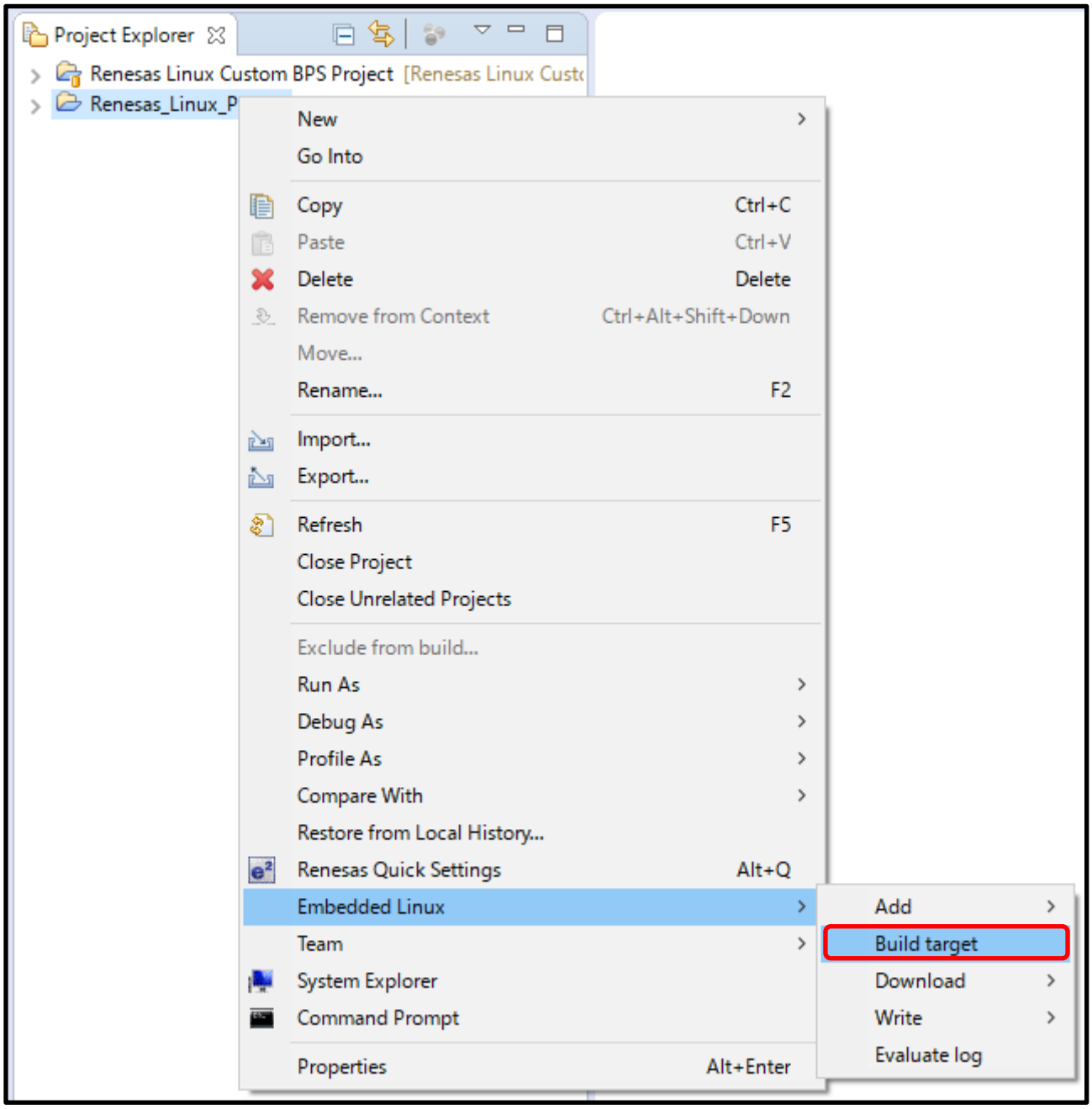
6. In addition, if you wish to add a Yocto recipe, select the [Optional components] tabbed page and click on the [Add] button to enter the function of the Yocto recipe (name of the recipe) to be added to the targets for building.
   To omit a function which you have entered from the targets for building, select the function and click on the [Delete] button.
   For details on how to add a Yocto recipe, see 3.11.3 Adding Yocto Recipes.



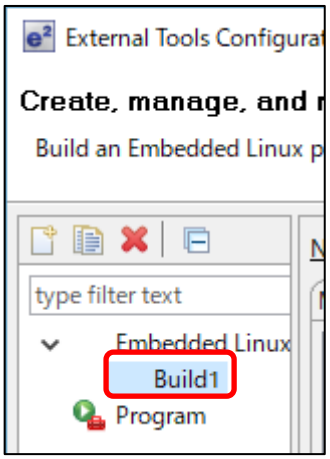
7. When the above setup processes are completely finished, click on the [Apply] button to save the setup.

3.8.2 Executing Build

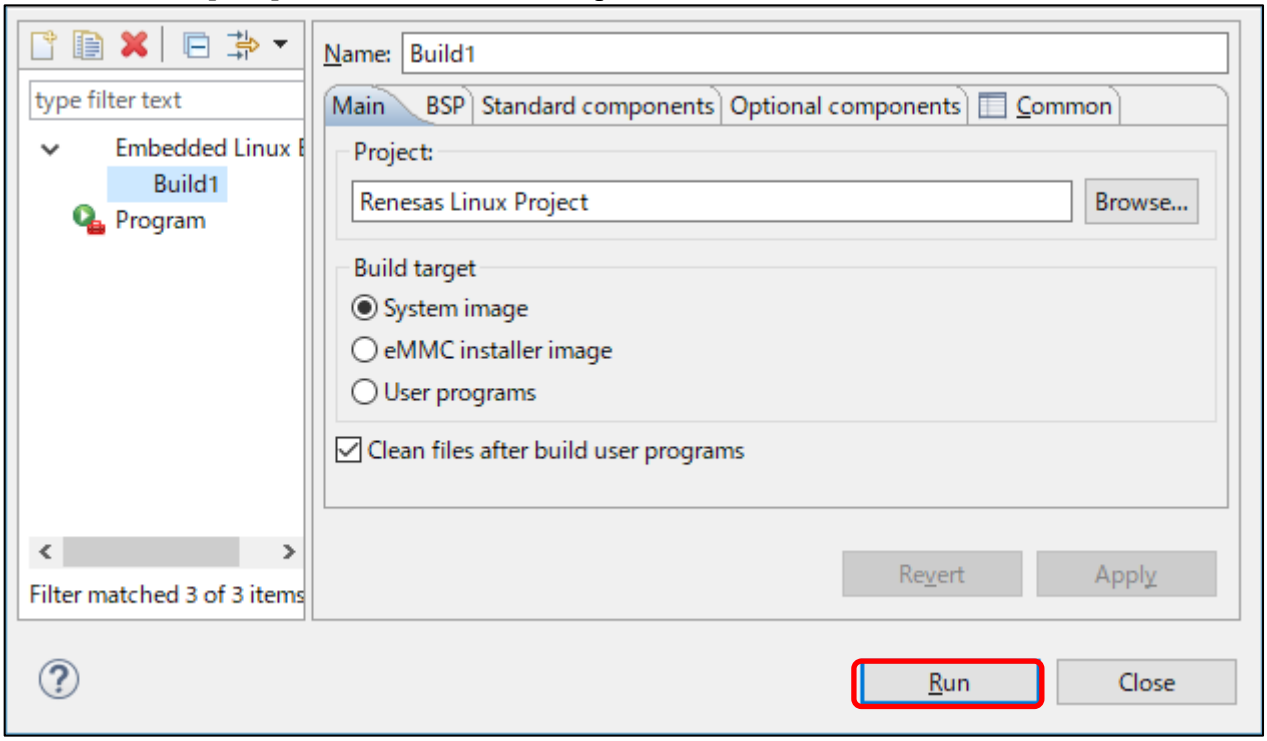
1. Right-click on the embedded Linux project in the [Project Explorer] pane and select [Embedded Linux] -> [Build target] from the pop-up menu.



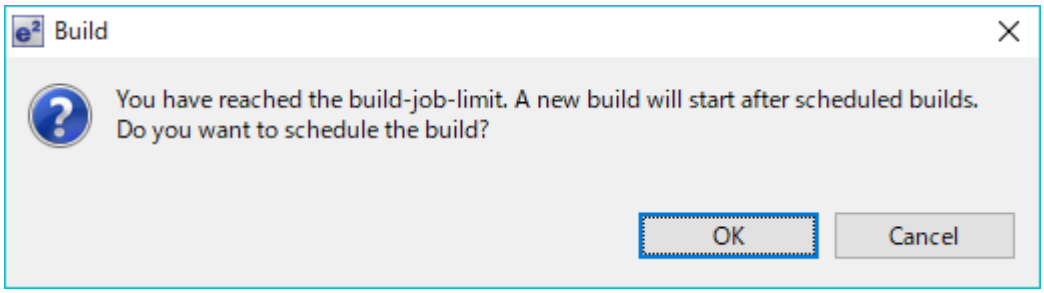
2. The [External Tools Configurations] window appears. Select the setup name that has been created in 4.8.1 Setting up Build.



3. Click on the [Run] button to start the build process.

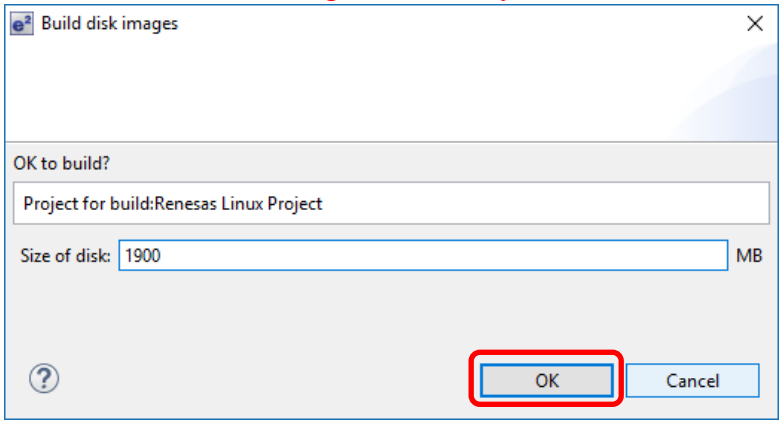


If other users (up to four) are already using the build server for build processing, the following dialog box appears.

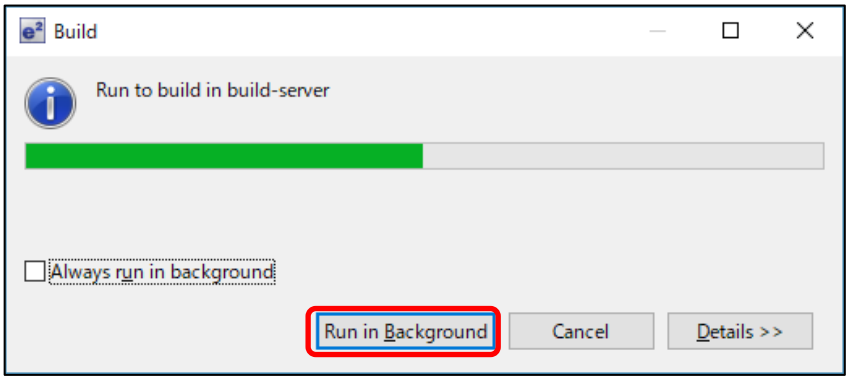


Clicking on the [OK] button executes the build process as soon as one of those for which another user is using the build server has been completed.
Clicking on the [Cancel] button stops execution of the build process.

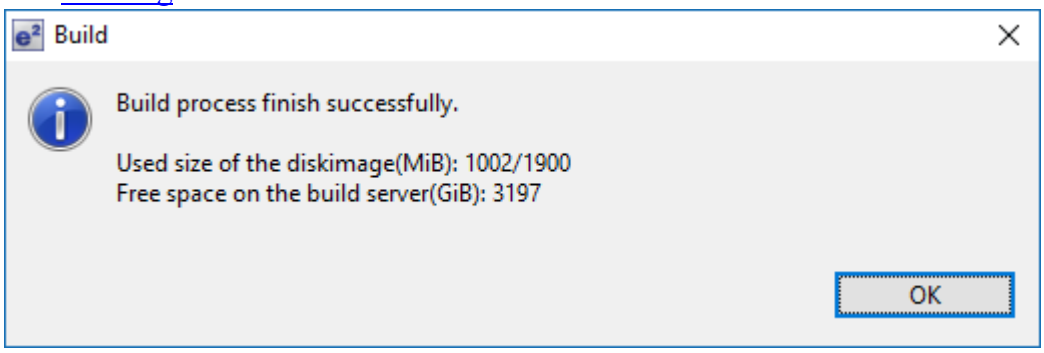
4. If [System image] or [eMMC installer image] has been selected as the build target during the build setup process, the window for setting the disk size appears.
   Note: When no change is necessary, use the default setting as it is.



5. When the build process starts, a progress bar appears. To allow the e^2 studio to do any other job during the build process, click on the [Run in Background] button.



6. On completion of building, the dialog box for the results of building appears, and the results of building are stored on the build server. In addition to the results of building, the dialog box for the results of building includes information on the size of the built rootfs and the amount of empty space on the HDD of the build server.
   After completion of building, the [Download image file] window opens automatically.
   Download the results of building to the client PC as described in 3.9.1 Downloading Results of Building.



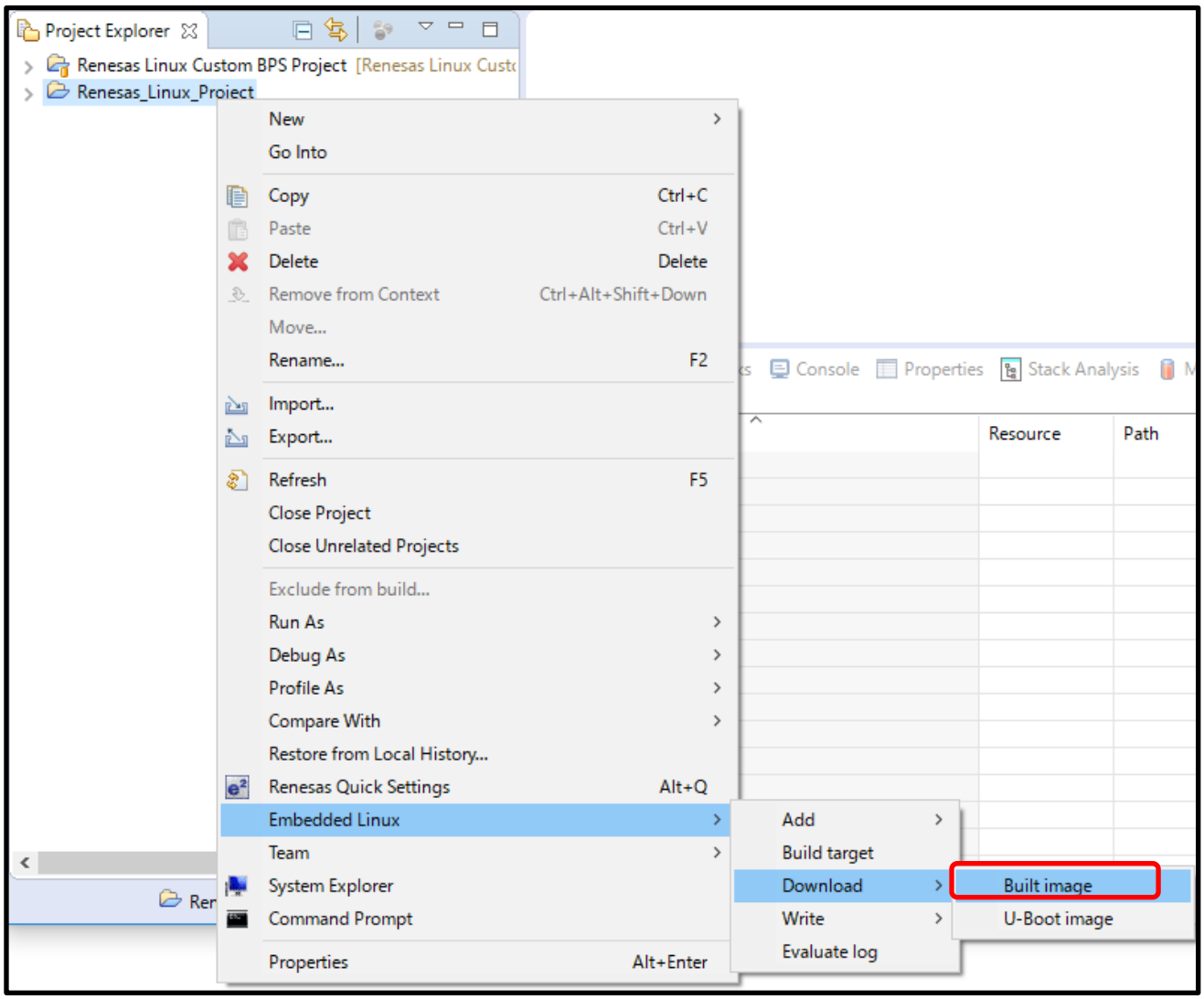
3.9 Downloading

This section describes the procedure for downloading the results of building on the build server and U-Boot binary files.

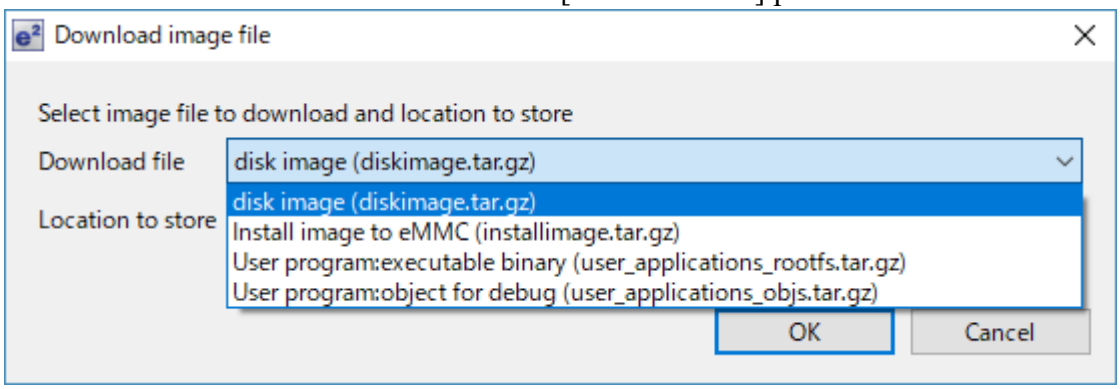
3.9.1 Downloading Results of Building

This subsection describes the procedure for downloading the results of building on the build server.

1. Right-click on the embedded Linux project in the [Project Explorer] pane and select [Embedded Linux] -> [Download] -> [Built image] from the pop-up menu.

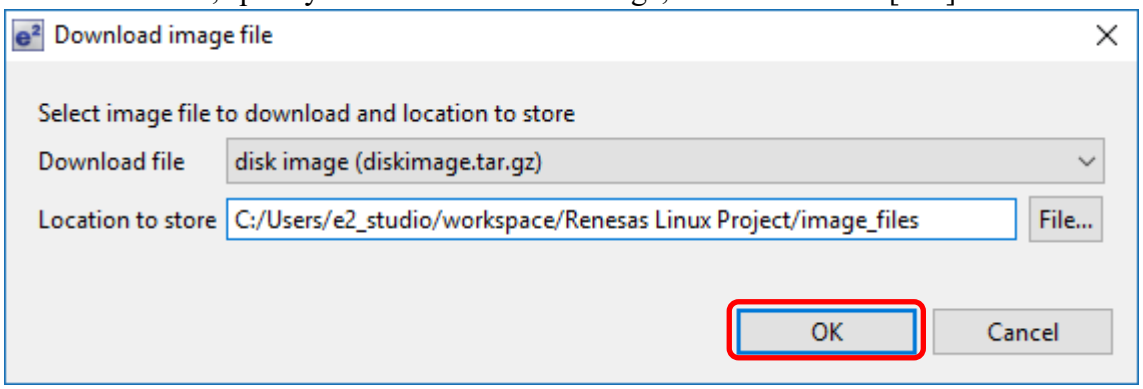


2. Select the file to be downloaded in the [Download file] pull-down menu.

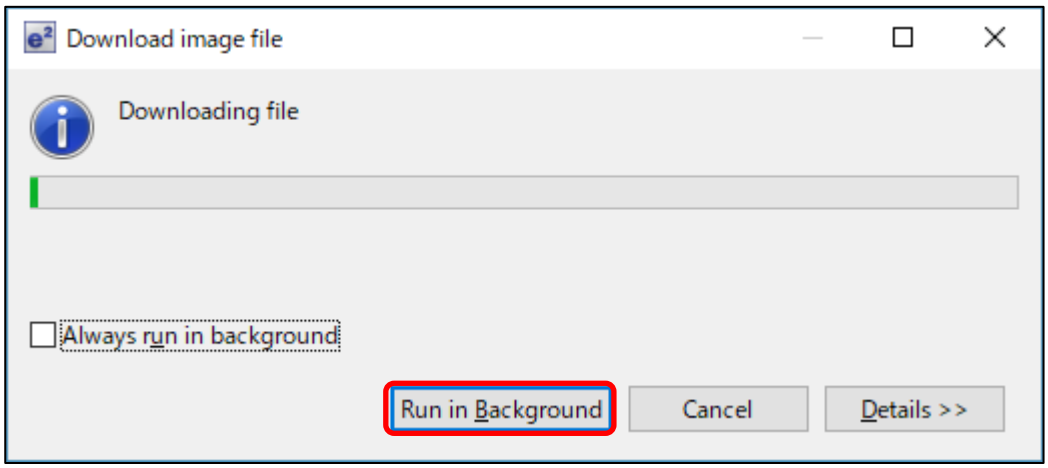


Parameter	Description of the File to Download
disk image (diskimage.tar.gz)	Image file used for booting from an SD card
Install image to eMMC (installimage.tar.gz)	Image file used for booting from an eMMC on the target board
User program:executable binary (user_applications_rootfs.tar.gz)	Binary file used for developing rootfs of the user program. Used for saving executable files such as user applications in the Linux environment of the target board that has been started up.
User program:object for debug (user_applications_objs.tar.gz)	A set of added user programs or objects

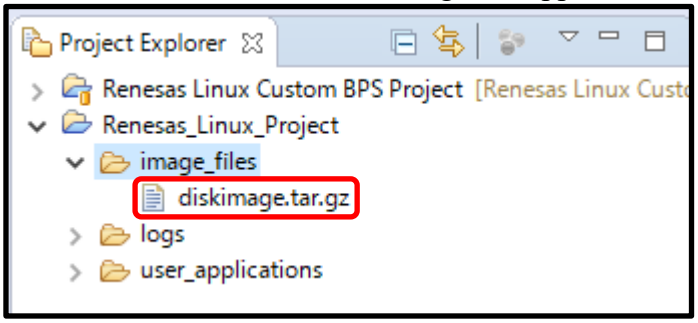
3. Select a file, specify the destination for storage, and click on the [OK] button.



4. When the downloading starts, a progress bar appears. To allow the e^2 studio to do any other job during the downloading process, click on the [Run in Background] button.



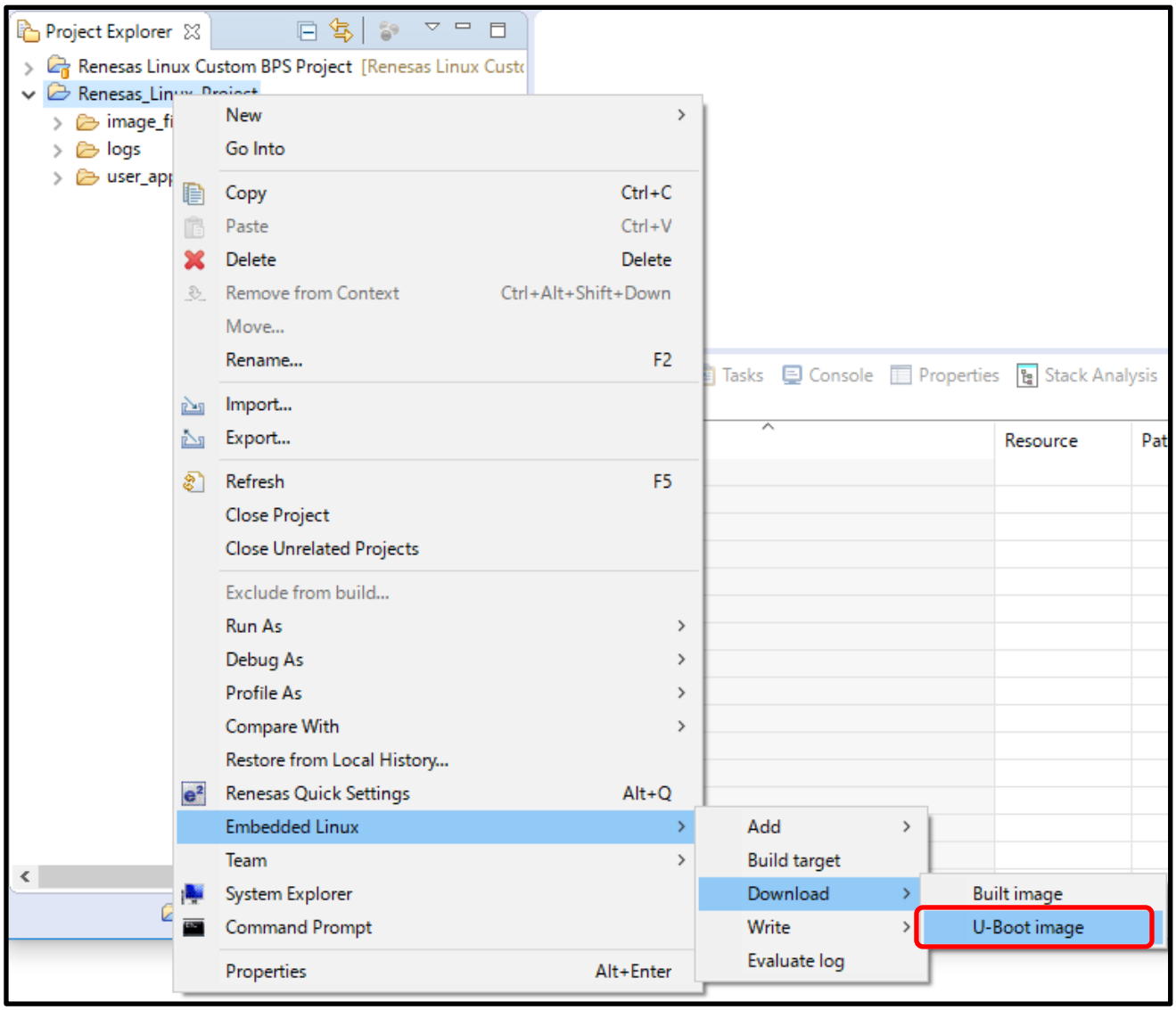
5. Confirm that the selected image file appears under [image_files] in the [Project Explore] pane.



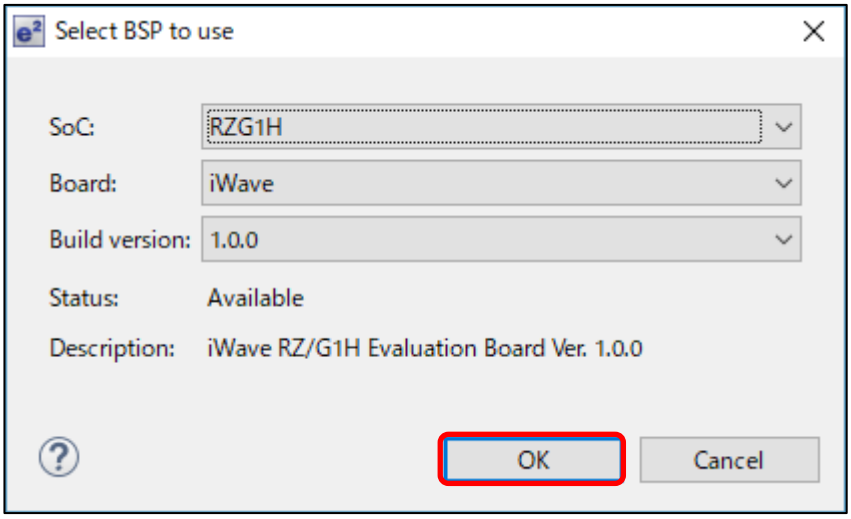
3.9.2 Downloading the U-Boot File

This subsection describes the procedure for downloading the U-Boot file, which is the boot loader for the target board.

1. Right-click on the embedded Linux project in the [Project Explorer] pane and select [Embedded Linux] -> [Download] -> [U-Boot image] from the pop-up menu.



2. Selecting the target board to be downloaded and clicking on the [OK] button starts downloading of the U-Boot file.



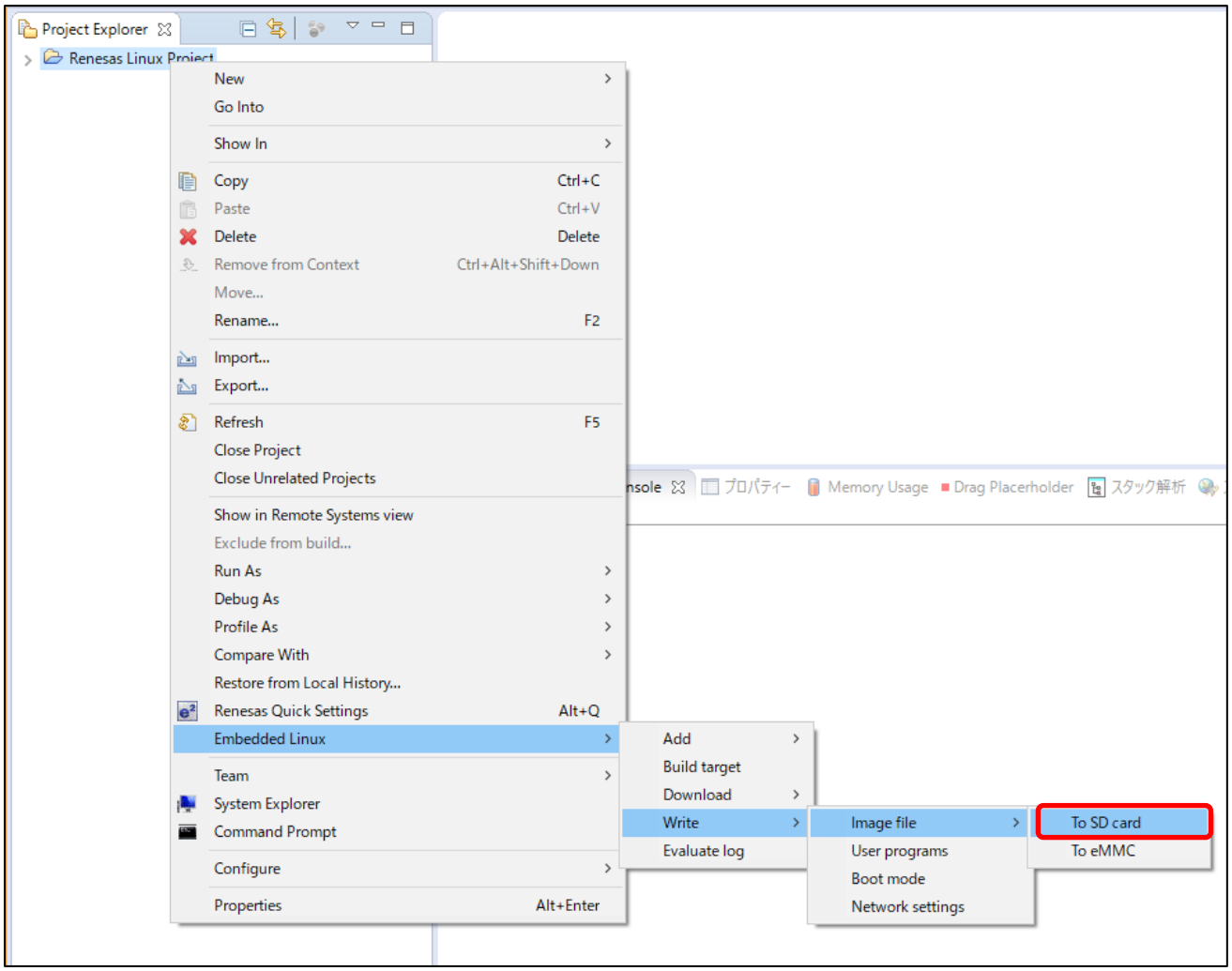
3.10 Writing

This section describes the procedure for writing the downloaded results of building to the target board, and the procedure for switching boot mode.

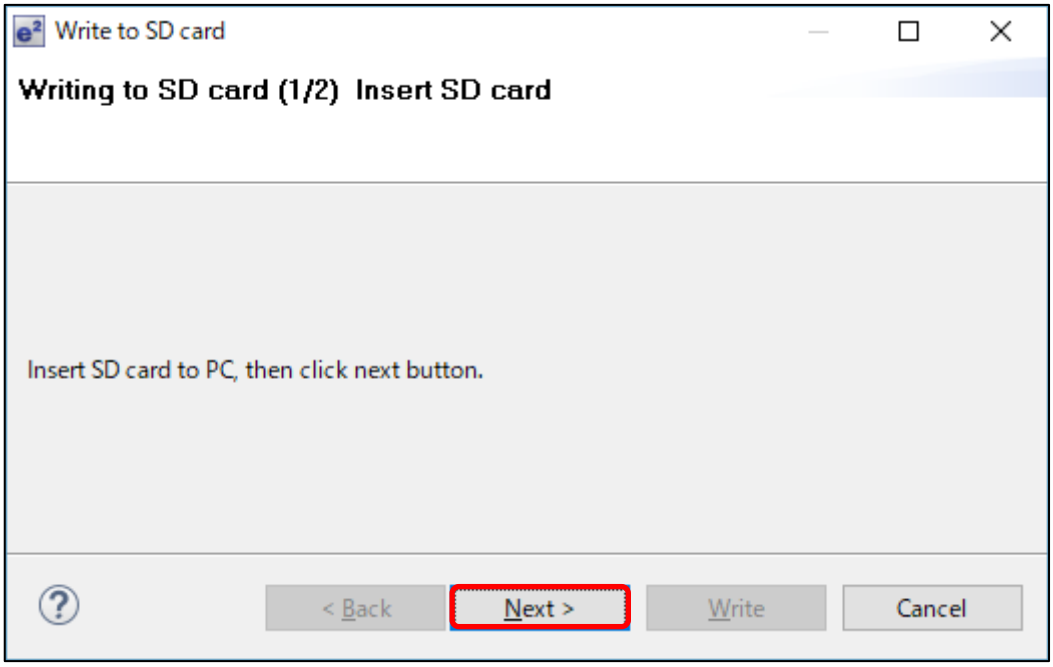
3.10.1 Writing to an SD Card

This subsection describes the procedure for writing an image file for SD card boot or eMMC installer to the SD card that was downloaded as described in 3.9.1 Downloading Results of Building.

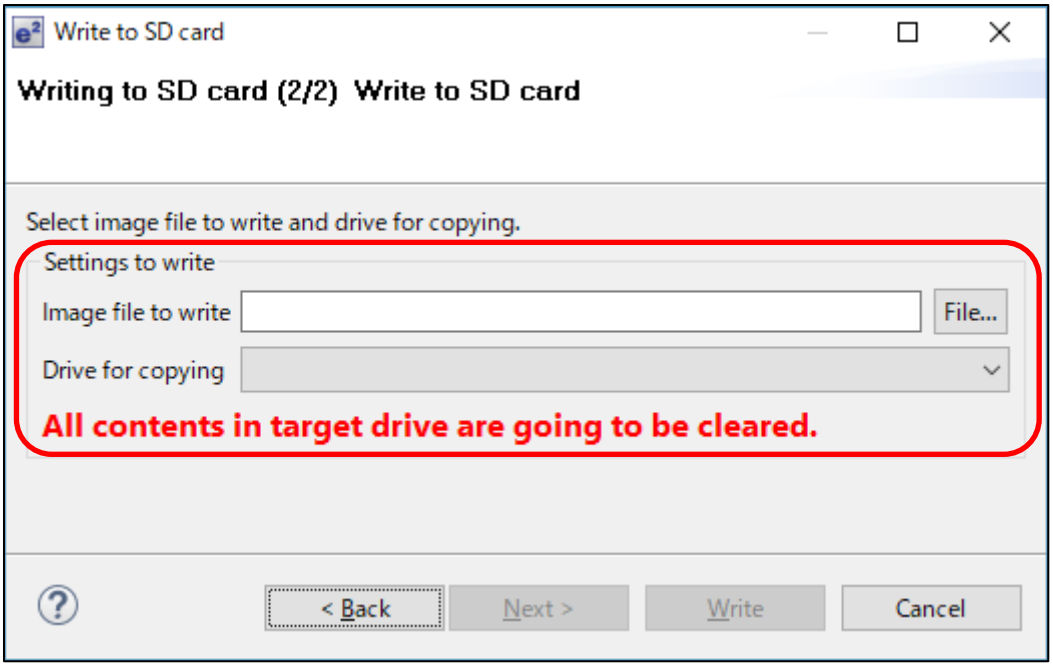
1. Right-click on the embedded Linux project in the [Project Explorer] pane and select [Embedded Linux] -> [Write] -> [Image file] -> [To SD card] from the pop-up menu.



2. Follow the direction in the window below to insert an SD card in the slot of the Client PC.



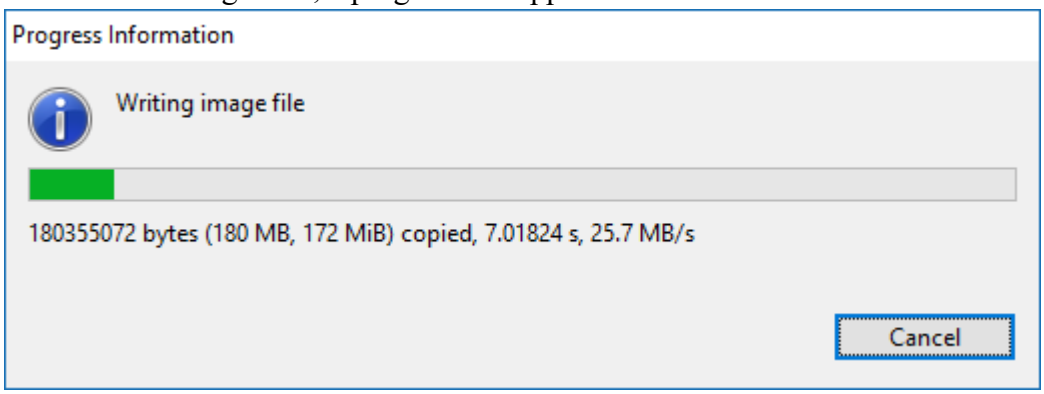
3. Next, click on the [File] button and specify the image file or eMMC installer to be written to SD card then specify a drive as the destination for copying the image file (SD card).



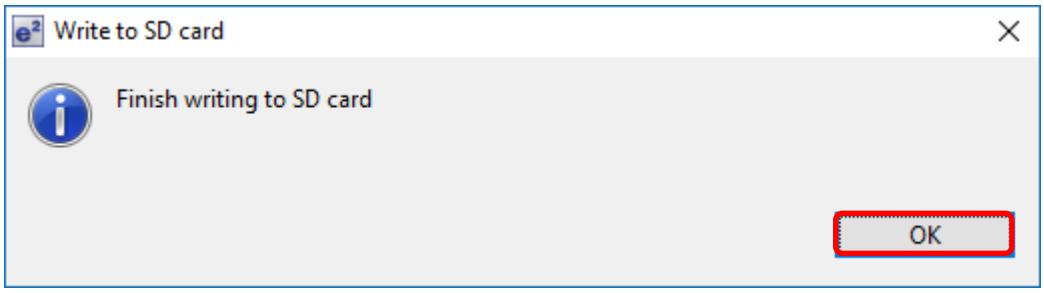
[Point for Caution on Writing to an SD Card]

All data in the drive specified as the destination for copying the image file are overwritten by the file copied in writing to the SD card. Be careful not to specify a drive containing a card that holds anything you need.

4. When writing starts, a progress bar appears.



5. When writing is finished, the completion window appears. Click on the [OK] button.

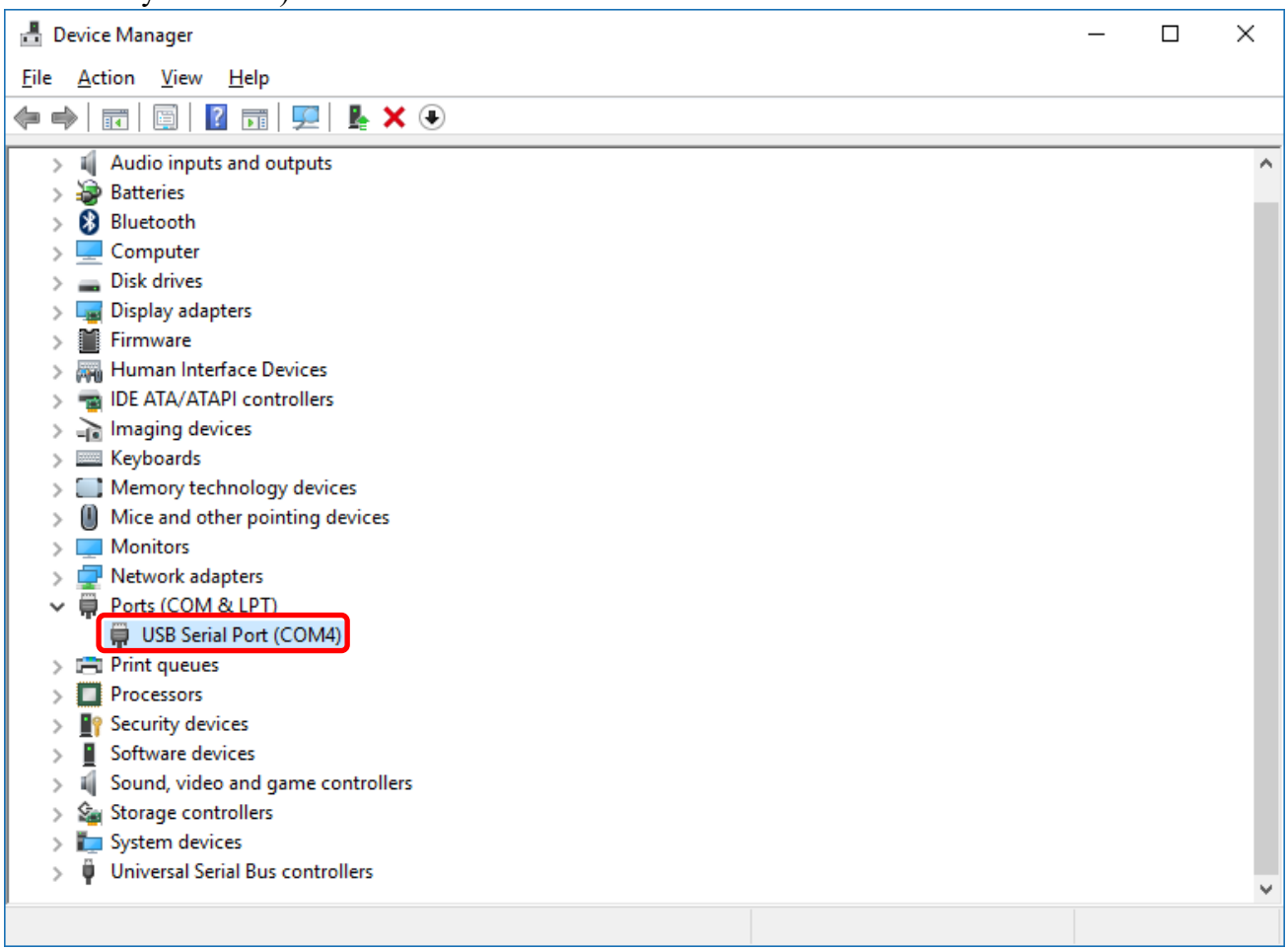


3.10.2 Writing to an eMMC

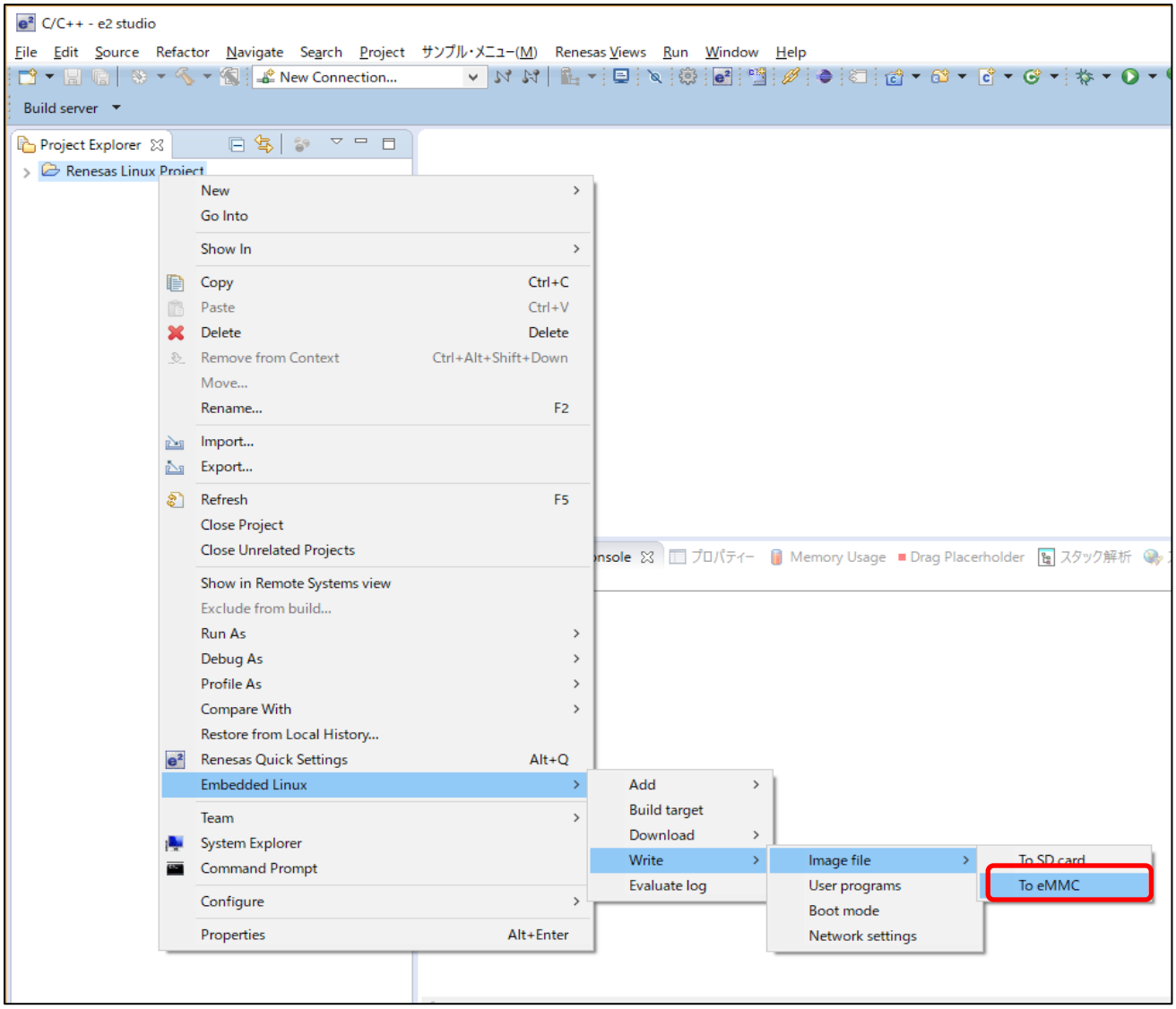
This subsection describes the procedure for writing an eMMC installation image downloaded in 3.9.1 Downloading Results of Building, to a target board.

Notes: 1. The eMMC installation image is written under the environment in which the SD card boot has been executed. Therefore, the SD card boot environment needs to be configured first. For how to configure the SD card boot environment, see 3.10.1 Writing to an SD Card.
2. Be sure to connect the client PC with the target board using a USB serial conversion cable.

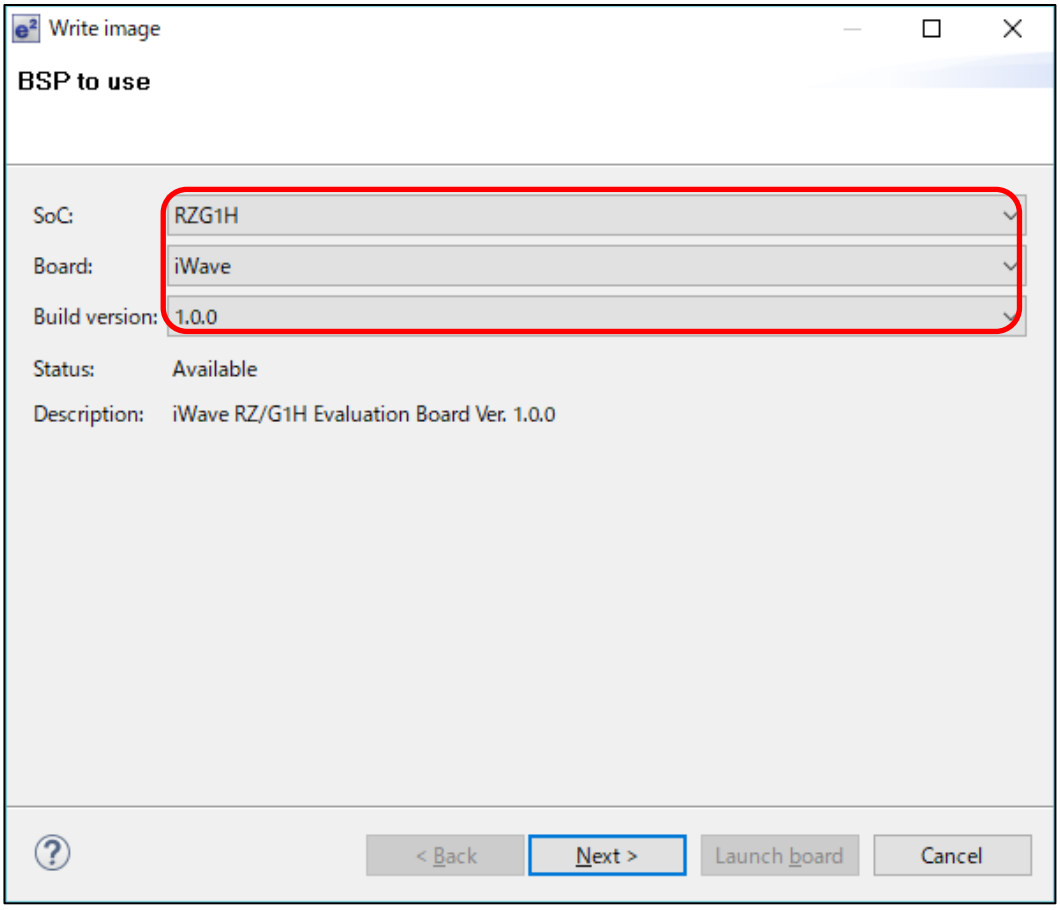
1. Start the Device Manager and confirm the presence of the COM port connected to the target board. (The following figure is shown as an example. Confirm the name of the USB serial driver actually installed.)



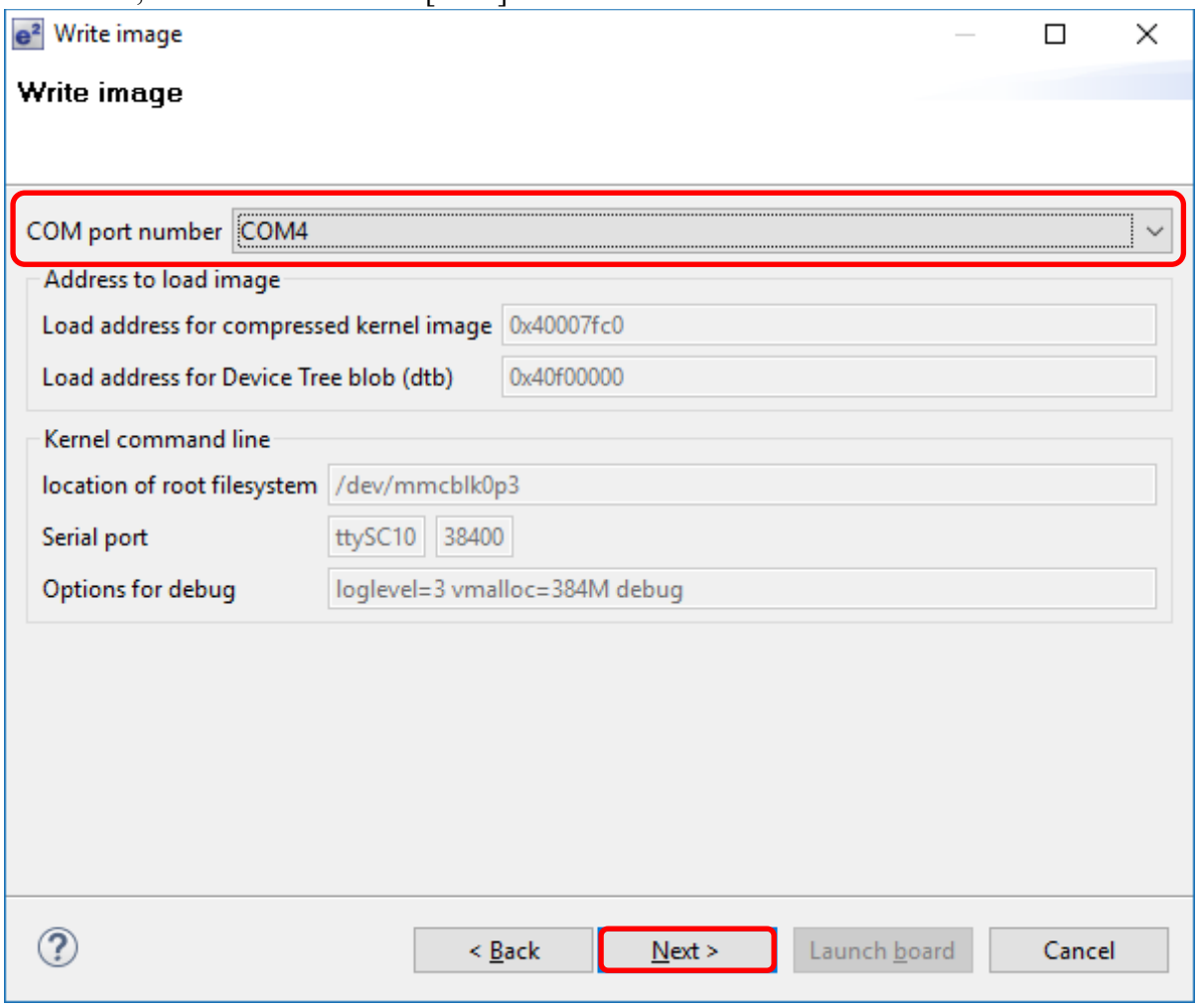
2. Right-click on the embedded Linux project in the [Project Explorer] pane and, select [Embedded Linux] -> [Write] > [Image file] > [To eMMC] from the pop-up menu.



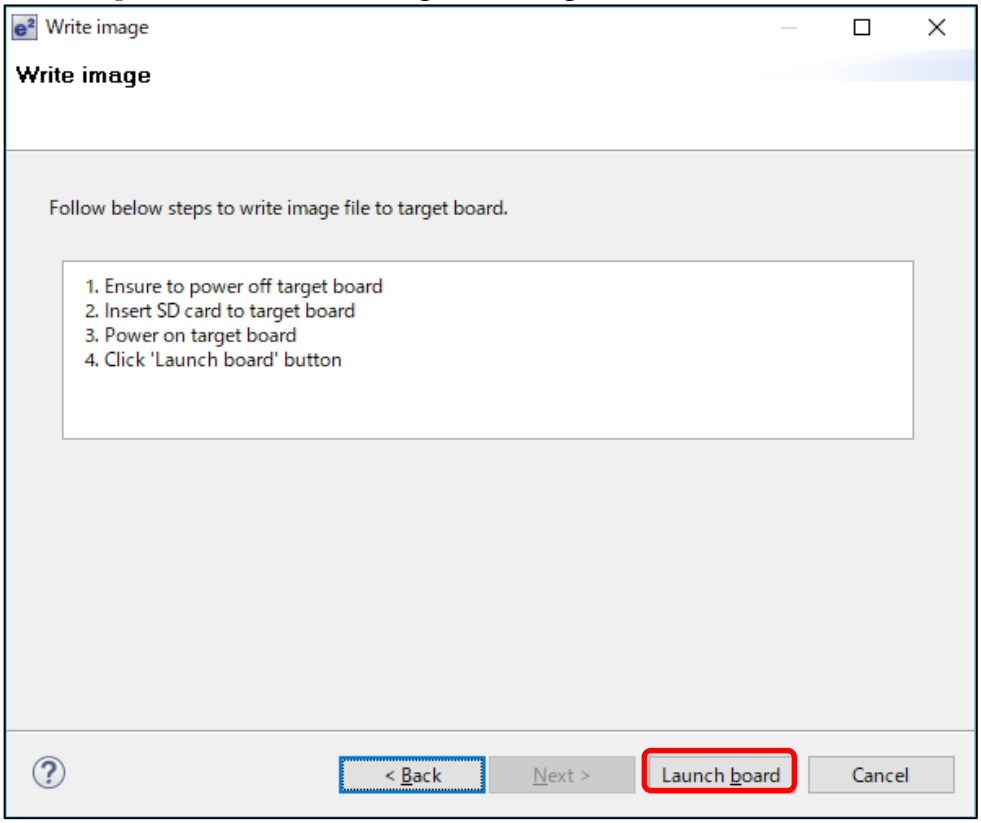
3. When the [Write image – BSP to use] window opens, specify the SoC, board, and build version.



4. The [Write image] window opens. Select the COM port for which the USB-serial driver was added, and then click on the [Next] button.



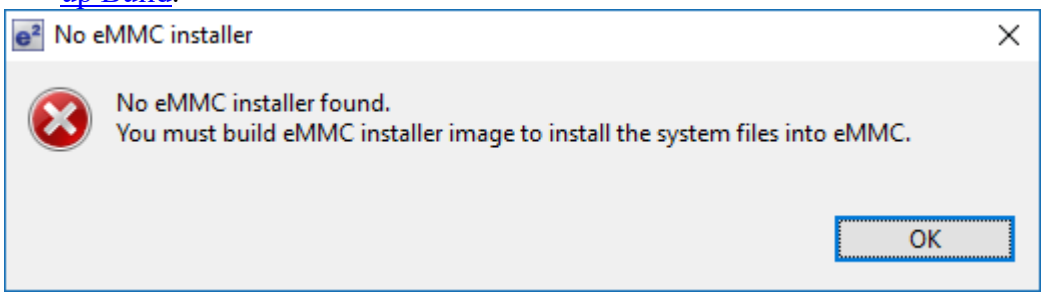
5. Follow the directions in the window. First confirm that the target board is turned off. Then insert the SD card in the slot of the target board, and turn on the target board. Clicking on the [Launch board] button executes writing of an image file to the eMMC.



6. To start booting from the eMMC after completion of writing, re-boot the target board. Here, the boot device mode needs to be switched to eMMC. For how to switch the boot mode to eMMC, see 3.10.4 Switching Boot Mode.

   If the eMMC installer is not present on the SD card during writing, the following error window appears.

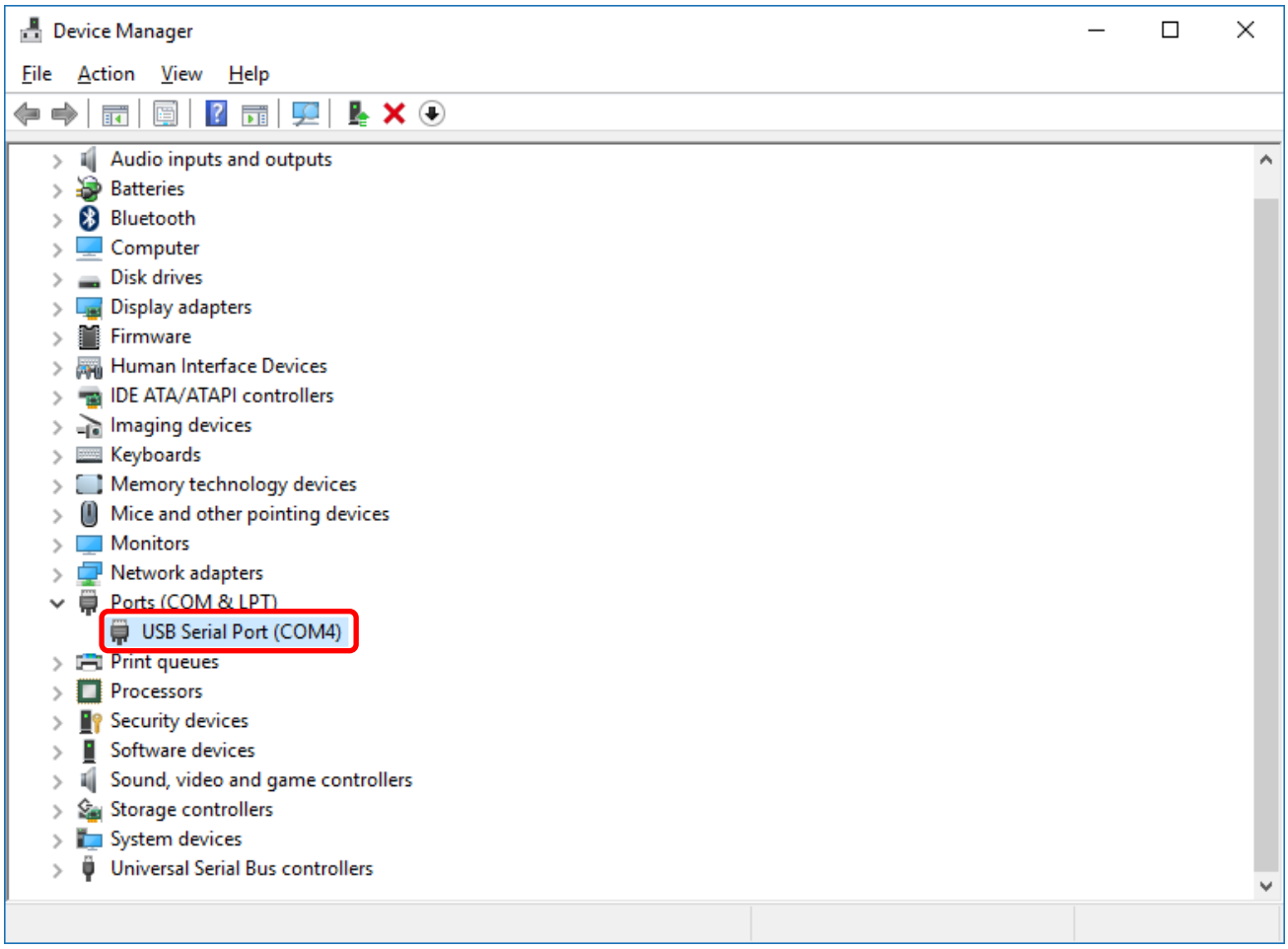
   When the error window appears, check whether an image file which includes the eMMC installer has been written to the SD card.
   For details on how to create an image file which includes the eMMC installer, see 3.8.1 Setting up Build.



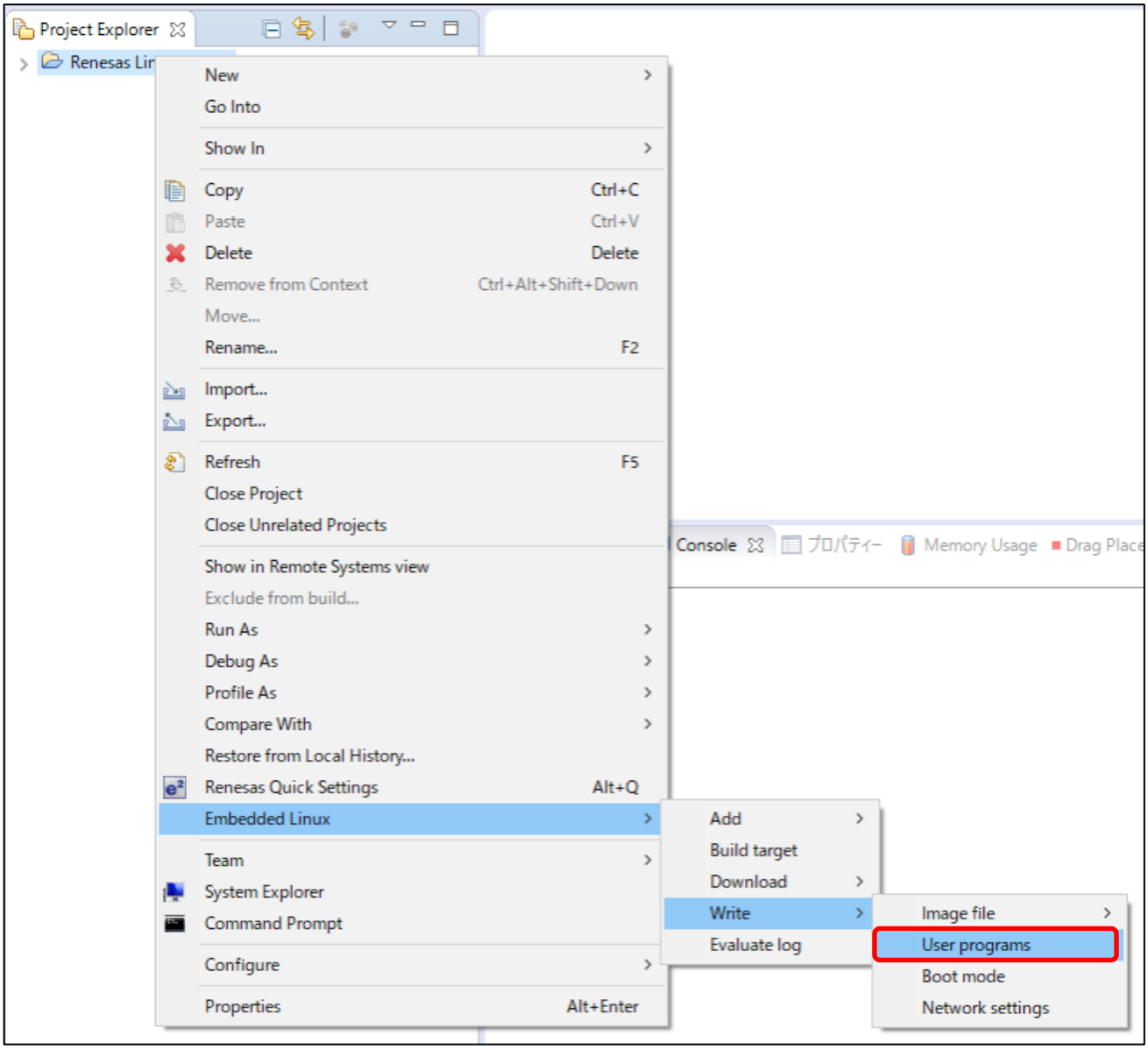
3.10.3 Writing a User Program

This subsection describes the procedure for writing an executable binary file of a user program downloaded in 3.9.1 Downloading Results of Building to the boot device (SD card or eMMC).
Note: Be sure to connect the client PC with the target board using a USB serial conversion cable.

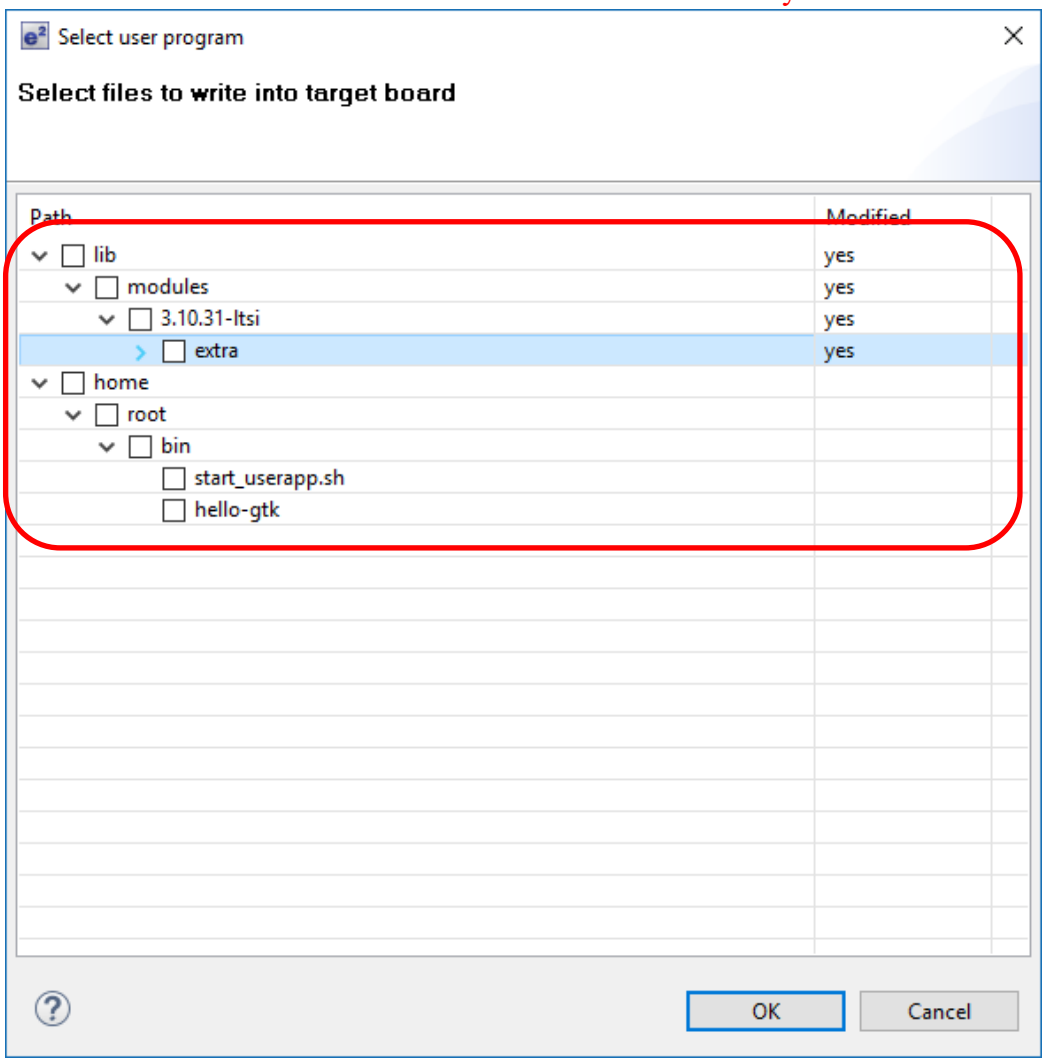
1. Start the Device Manager and confirm the presence of the COM port connected to the target board. (The following figure is shown as an example. Confirm the name of the USB serial driver actually installed.)



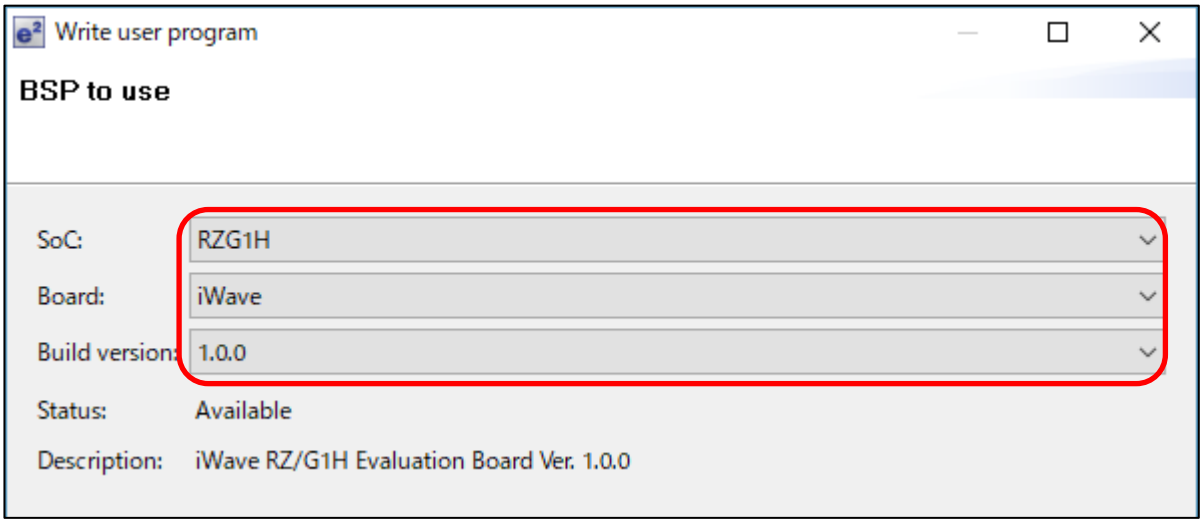
2. Right-click on the embedded Linux project in the [Project Explorer] pane and select [Embedded Linux] -> [Write] > [User programs] from the pop-up menu.



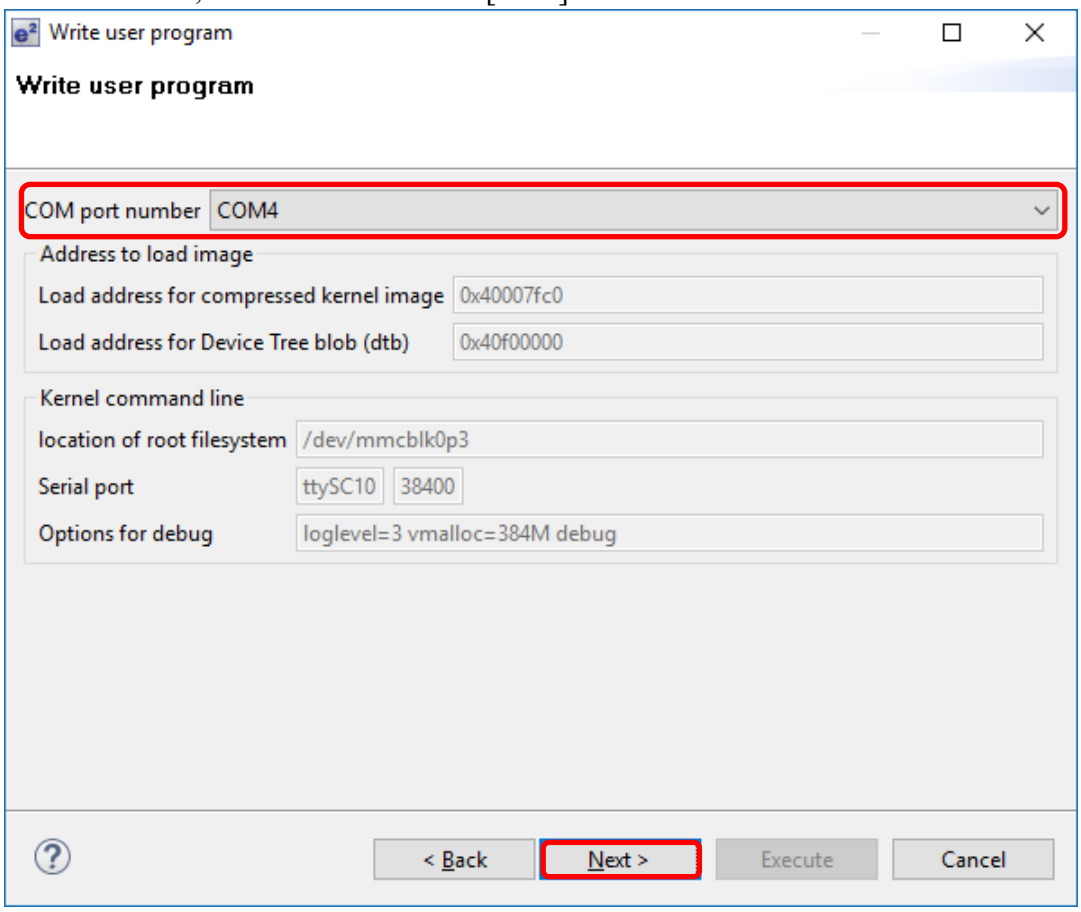
3. The window for selecting a user program opens. Place a check-mark against the name of the file to be written and click on the [OK] button.
   Notes: 1. The list is displayed in the form of a directory configuration; clicking on the [>] button of a directory displays the directories and files under that directory.
   2. For newly built files or files with differences from those produced in the last round of building, "yes" is displayed in the [Modified] column for the given files and their checkboxes are checked as files to be written by default.



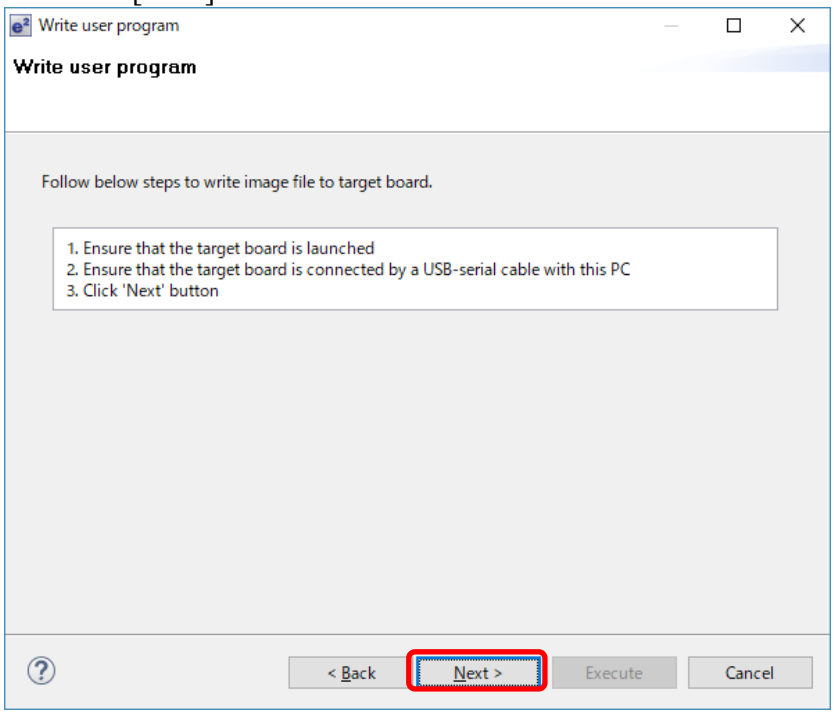
4. When the [Write user program – BSP to use] window opens, specify the SoC, board, and Build version.



5. The [Write user program] window opens. Select the COM port for which the USB-serial driver was added, and then click on the [Next] button.

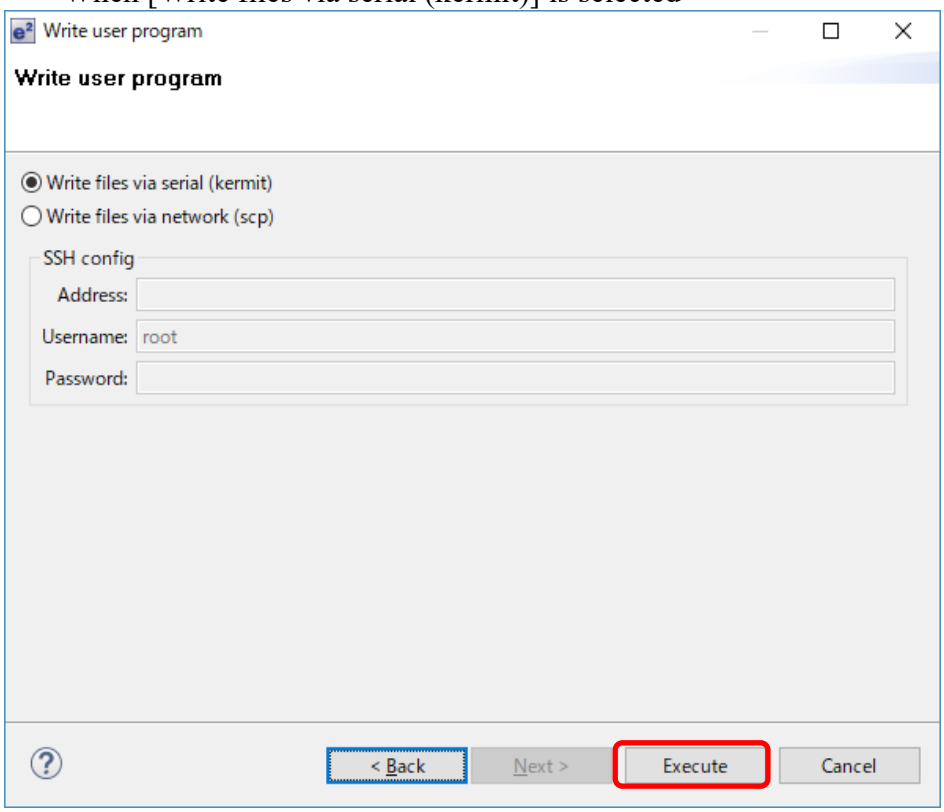


6. Follow the directions in the window. Start by confirming that the target board is running. Next, confirm that the PC and target board are connected by a USB-serial cable, and then click on the [Next] button.

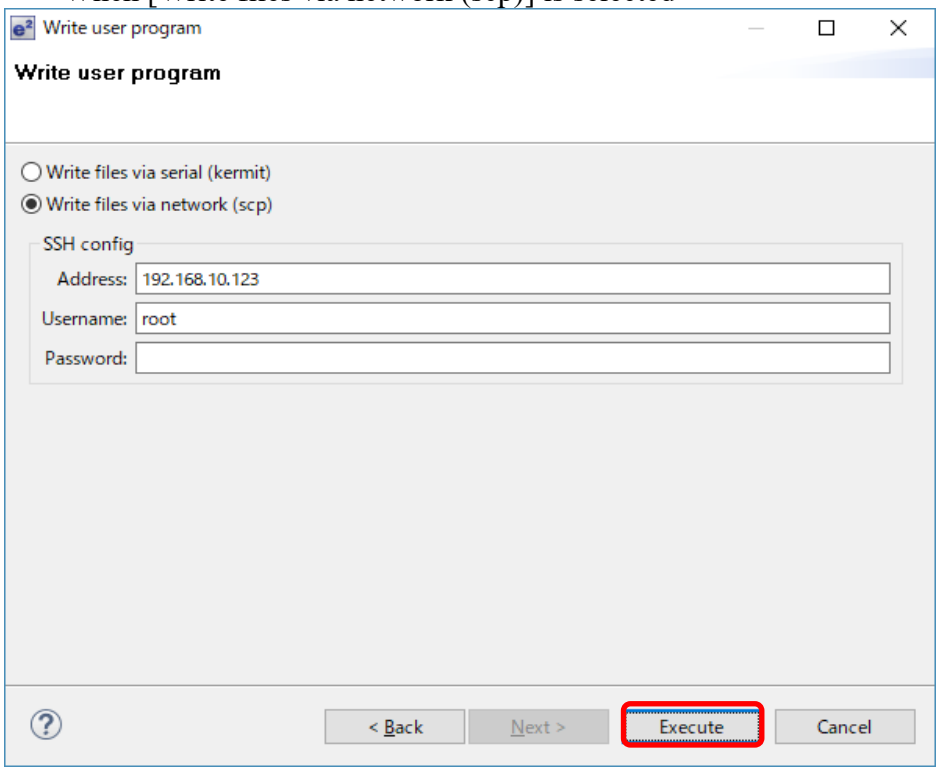


7. Select [Write files via serial (kermit)] or [Write files via network (scp)] as the method of writing the user program and click on the [Launch board] button. Writing of the user program to the target board will then start.
Note: For details on how to make the settings for the network when [Write files via network (scp)] is selected, see 3.10.5 Writing the Network Information.

<When [Write files via serial (kermit)] is selected>



<When [Write files via network (scp)] is selected>



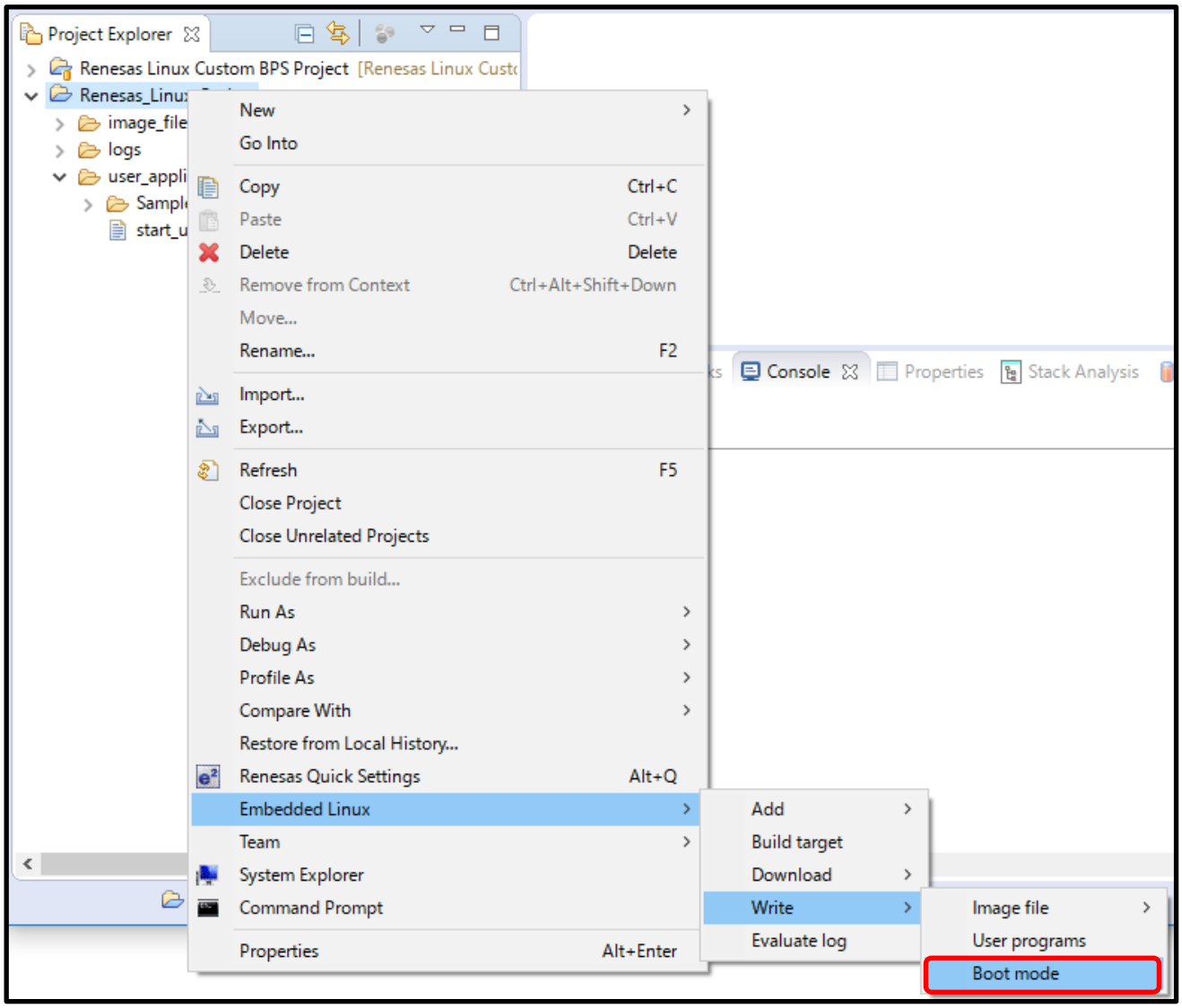
After writing is complete, turn off and re-boot the target board.

3.10.4 Switching Boot Mode

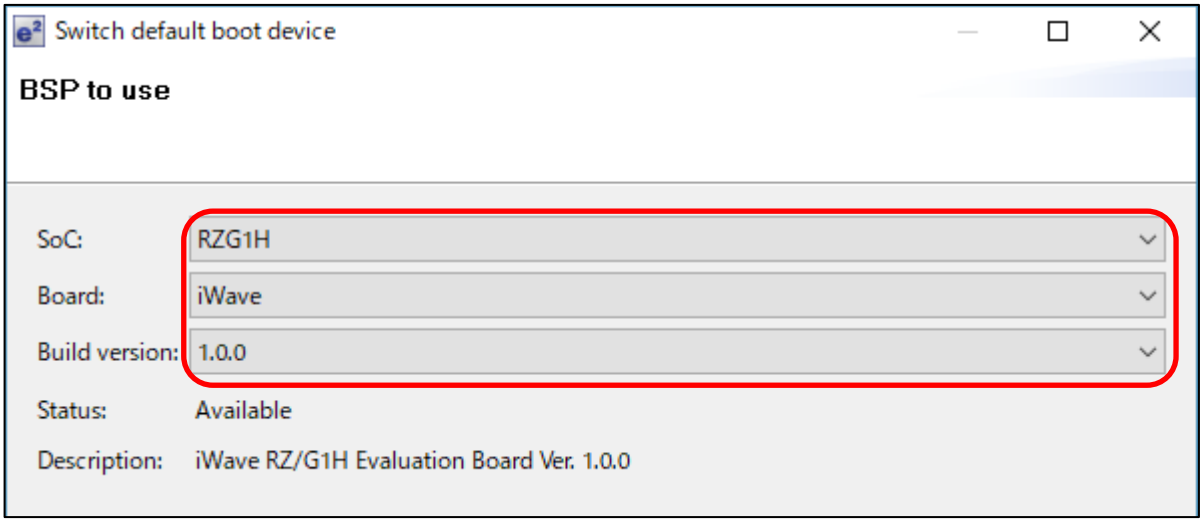
This subsection describes the procedure for switching the boot device (SD card or eMMC) for the target board.

Note: Set the serial ports in advance as described in 3.6.1 Setting Serial Ports. Be sure to connect the client PC with the target board using a USB serial conversion cable.

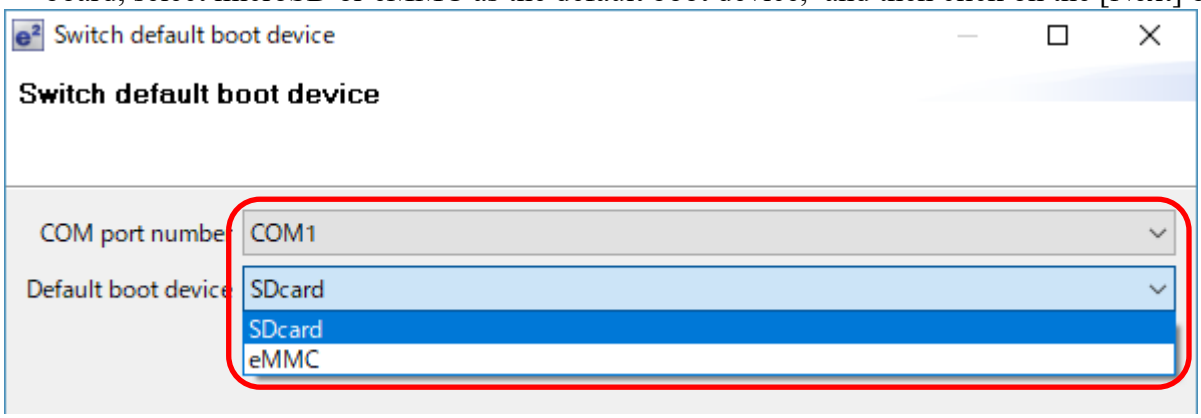
1. Right-click on the embedded Linux project in the [Project Explorer] pane and select [Embedded Linux] -> [Write] -> [Boot mode] from the pop-up menu.



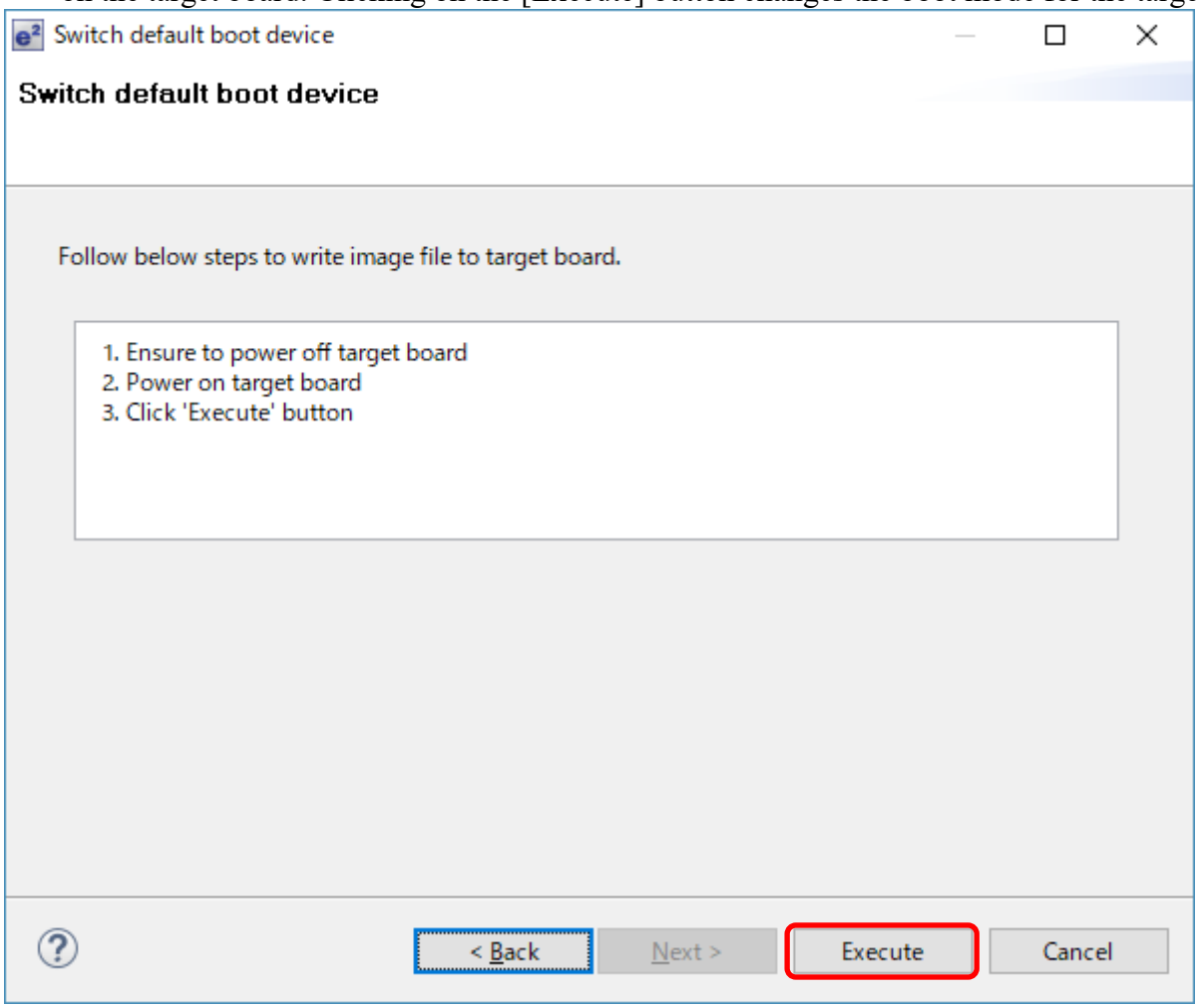
2. When the [Switch default boot device – BSP to use] window opens, specify the Soc, board, and build version.



3. The [Switch default boot device] window opens. Select the COM port connected to the target board, select microSD or eMMC as the default boot device, and then click on the [Next] button.



4. Follow the directions in the window. First confirm that the target board is turned off. Then turn on the target board. Clicking on the [Execute] button changes the boot mode for the target board.

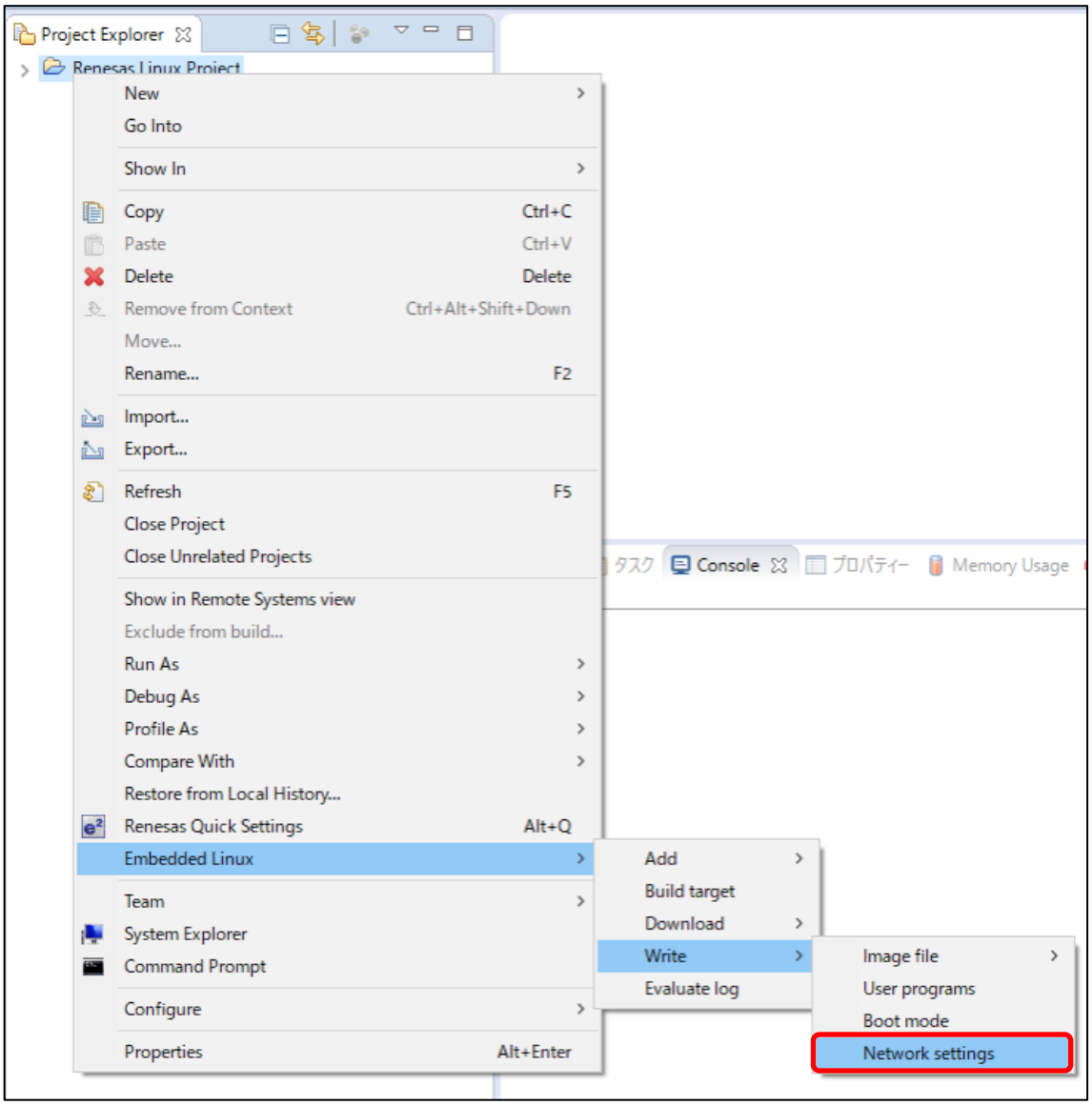


3.10.5 Writing the Network Information

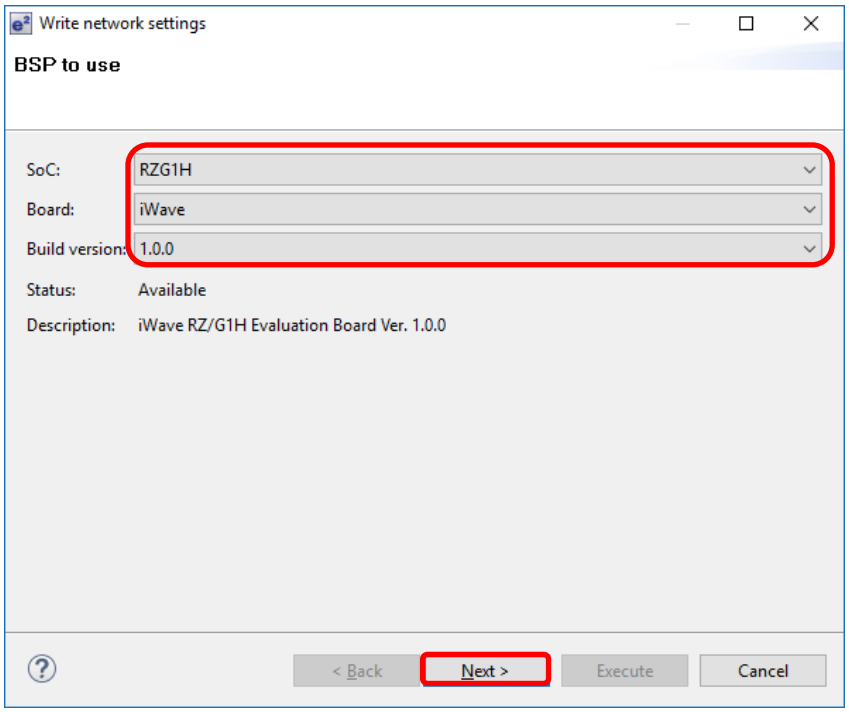
This subsection describes how to change the settings of the network parameters after starting up the target board.

Note: In the initial state, the values that were set in 3.6.2 Making the Network Settings are used.

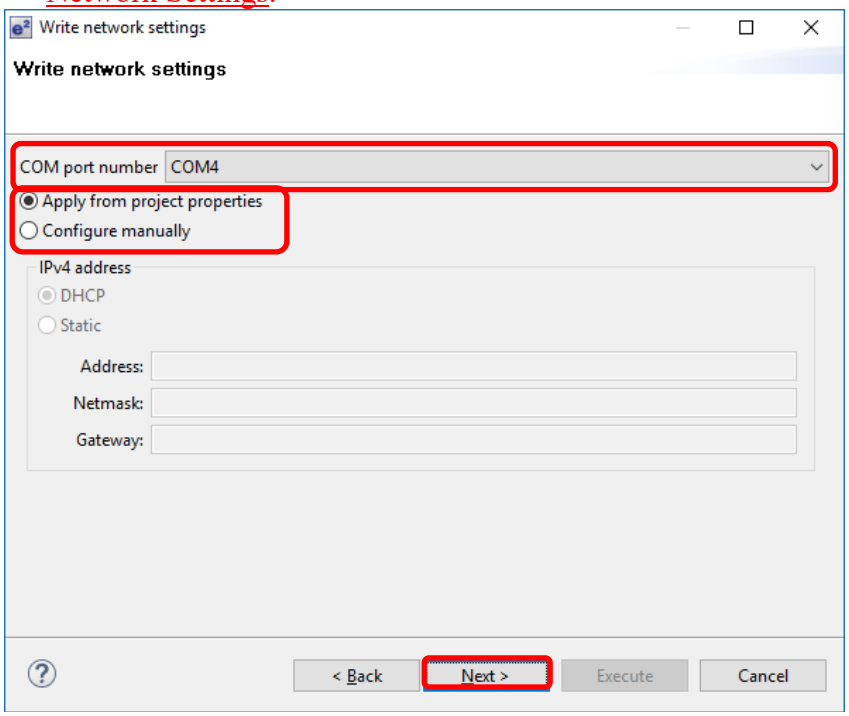
1. Right-click on the embedded Linux project in the [Project Explorer] pane and select [Embedded Linux] -> [Write] -> [Network settings] from the pop-up menu.



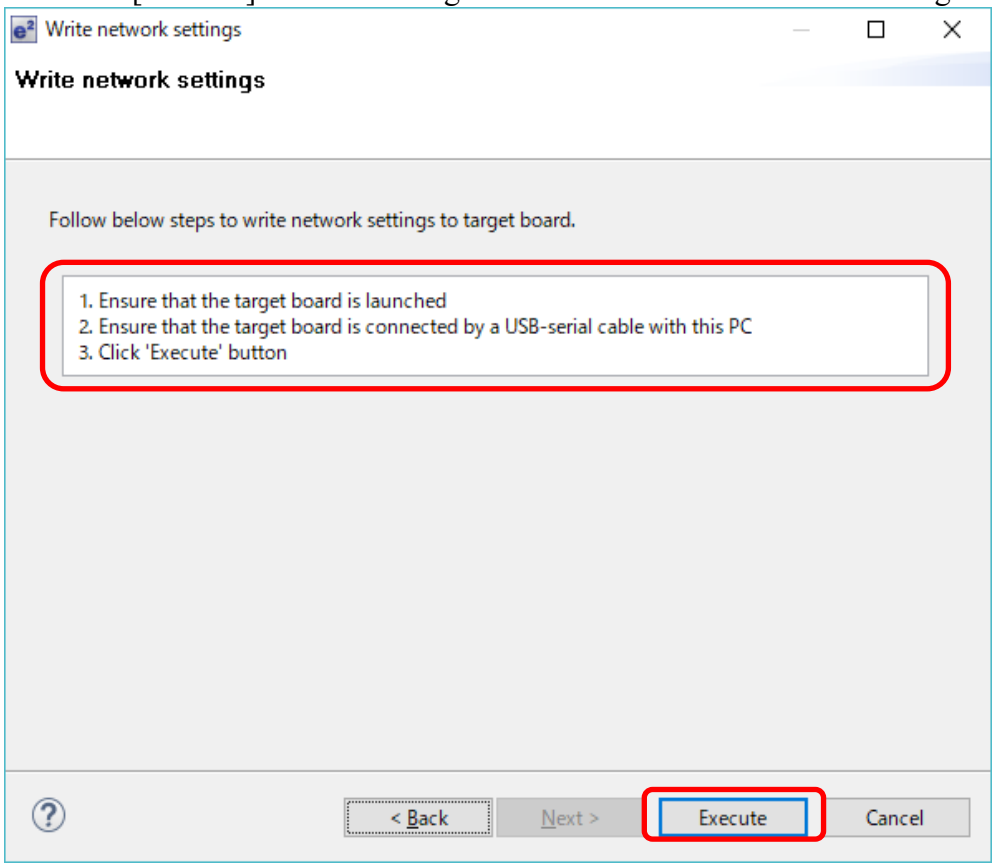
2. The [BSP to use] window appears. Select the target board to be used and click on the [Next] button.



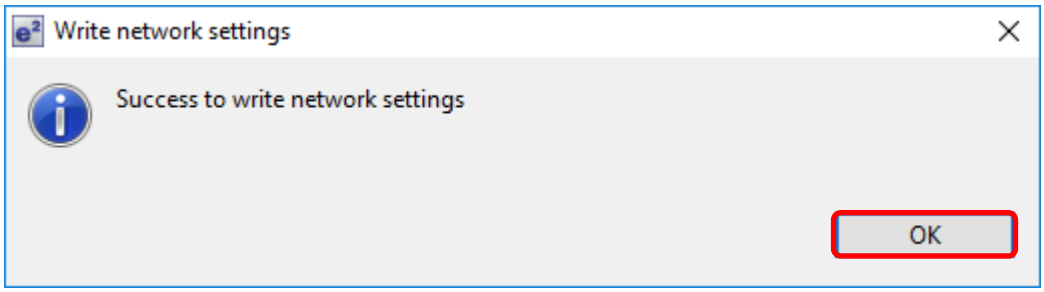
3. The [Write network settings] window appears. Select the COM port with which the target board is connected in the [COM port number] section, select [Apply from project properties] or [Configure manually], and click on the [Next] button.
Note: Selecting [Apply from project properties] reflects the settings made in 3.6.2 Making the Network Settings.



4. Follow the directions in the window. Start by confirming that the target board is running. Next, confirm that the PC and target board are connected by a USB-serial cable, and then click on the [Execute] button. Writing of the network information to the target board will then start.



5. When writing of the network information is finished, the completion window appears. Click on the [OK] button.



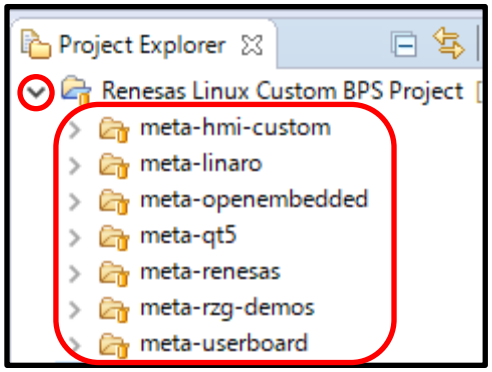
3.11 Customizing Yocto Recipes

This section describes the procedure for customizing the Yocto that has been downloaded in creating the embedded Linux custom BSP project.

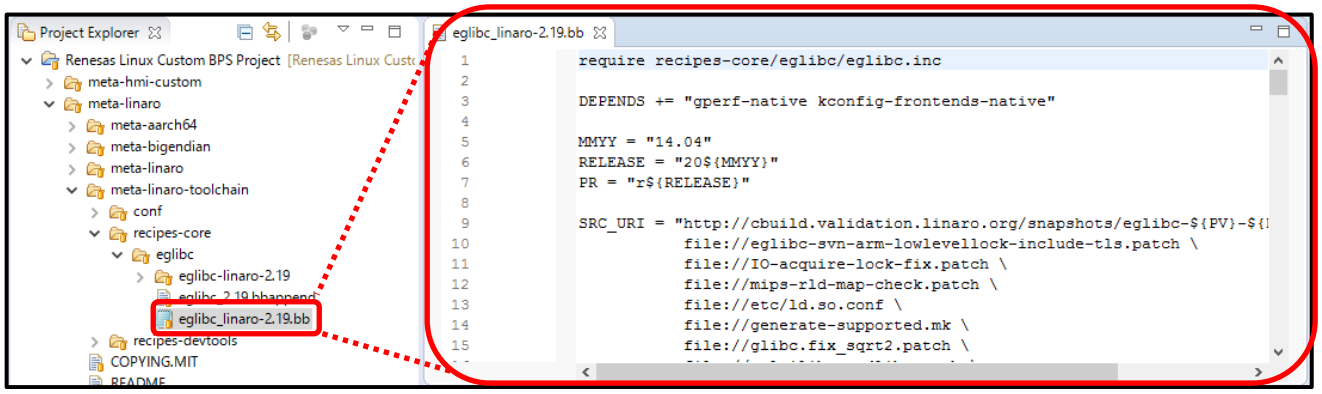
3.11.1 Editing Yocto Recipes

This subsection describes how to edit the Yocto recipe.

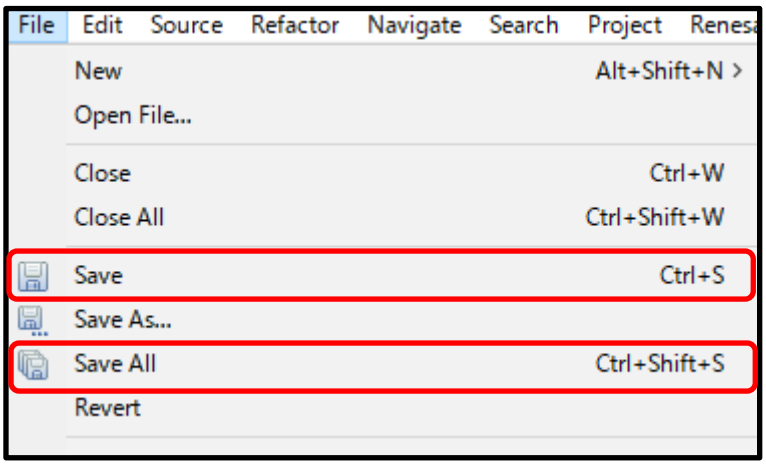
1. Open the embedded Linux custom BSP project that has been created to display the Yocto recipe directories.

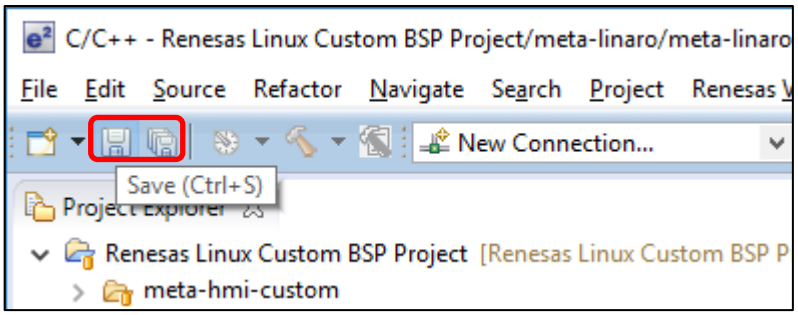


2. Double-click on or drag and drop the Yocto recipe to be edited to the editor window on the e[2] studio screen, and open and edit the file.



3. Select [Save] or [Save All] from [File] of the menu bar at the top of the window or from the icons at the top of the window to save the contents of editing.





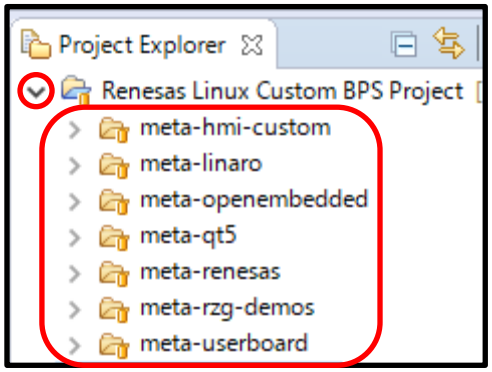
4. To build from the results of editing, open the [BSP] tabbed page in the window for setting up the build process and select [Custom BSP project on the workspace:], and proceed with the build process.

5. On the target board, confirm the operation of the image file which was built in step 4. If there are no problems with the operation, register the results of editing with the Git repository. For how to register, see 3.14.2 Registering with the Git.

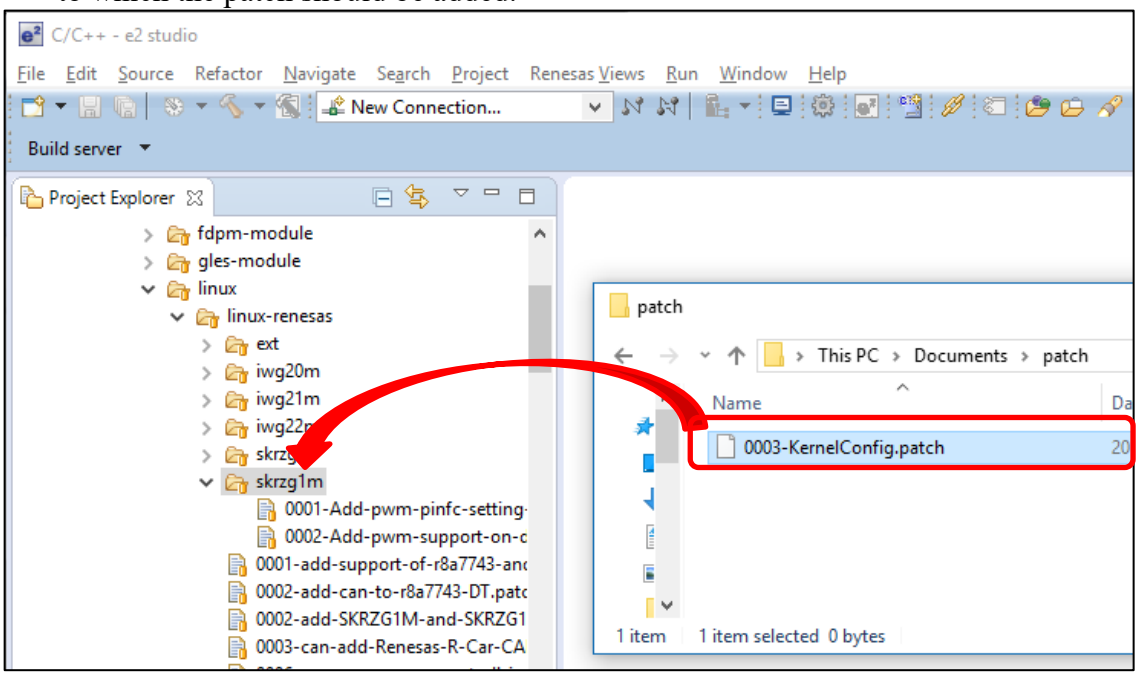
3.11.2 Adding Yocto Recipe Patches

This subsection describes how to add a Yocto recipe patch.

1. Open the embedded Linux custom BSP project that has been created to display the Yocto recipe directories.

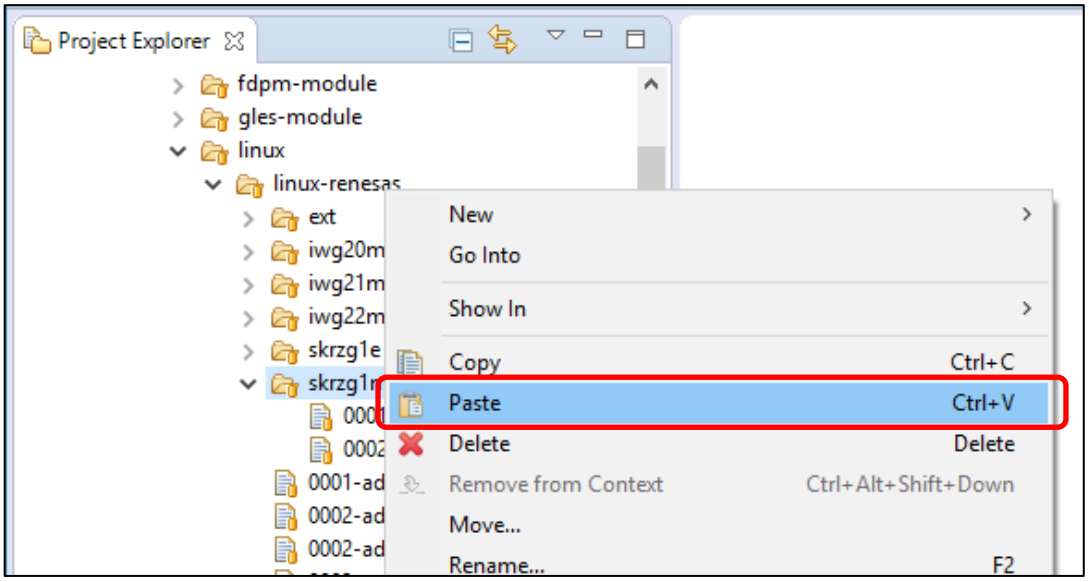


2. Drag and drop the Yocto recipe patch prepared on the client PC onto the Yocto recipe directory to which the patch should be added.



When the drag-and-drop operation in the e^2 studio is not possible, take the following steps to add the patch.

(1) Right-click on the Yocto recipe patch file to be added in Windows File Explorer and select [Copy] from the pop-up menu.
(2) Right-click on the folder to which the Yocto recipe patch is to be added in the [Project Explorer] pane of the e^2 studio and select [Paste] from the pop-up menu.



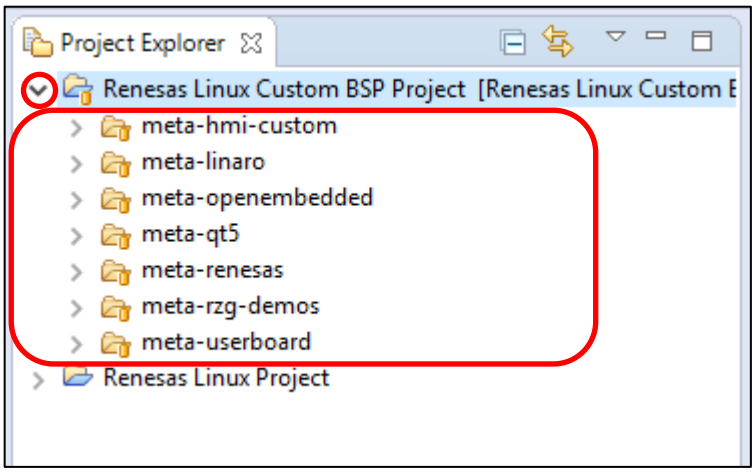
3. To build from the added patch, open the [BSP] tabbed page in the window for setting up the build process and select [Custom BSP project on the workspace:], and proceed with the build process.

4. On the target board, confirm the operation of the image file which was built in step 3. If there are no problems with the operation, register the added patch with the Git repository. For how to register, see 3.14.2 Registering with the Git.

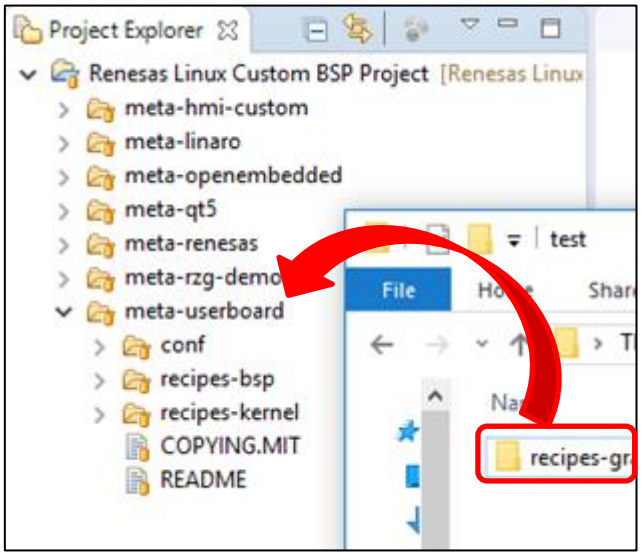
3.11.3 Adding Yocto Recipes

This subsection describes how to add a Yocto recipe.

1. Open the embedded Linux custom BSP project that has been created to display the Yocto recipe directories.

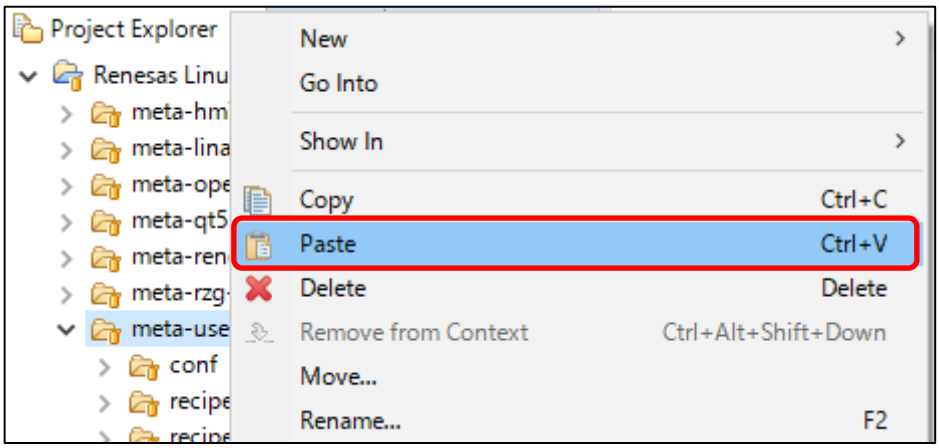


2. Drag and drop the Yocto recipe prepared on the client PC onto the meta-userboard directory.

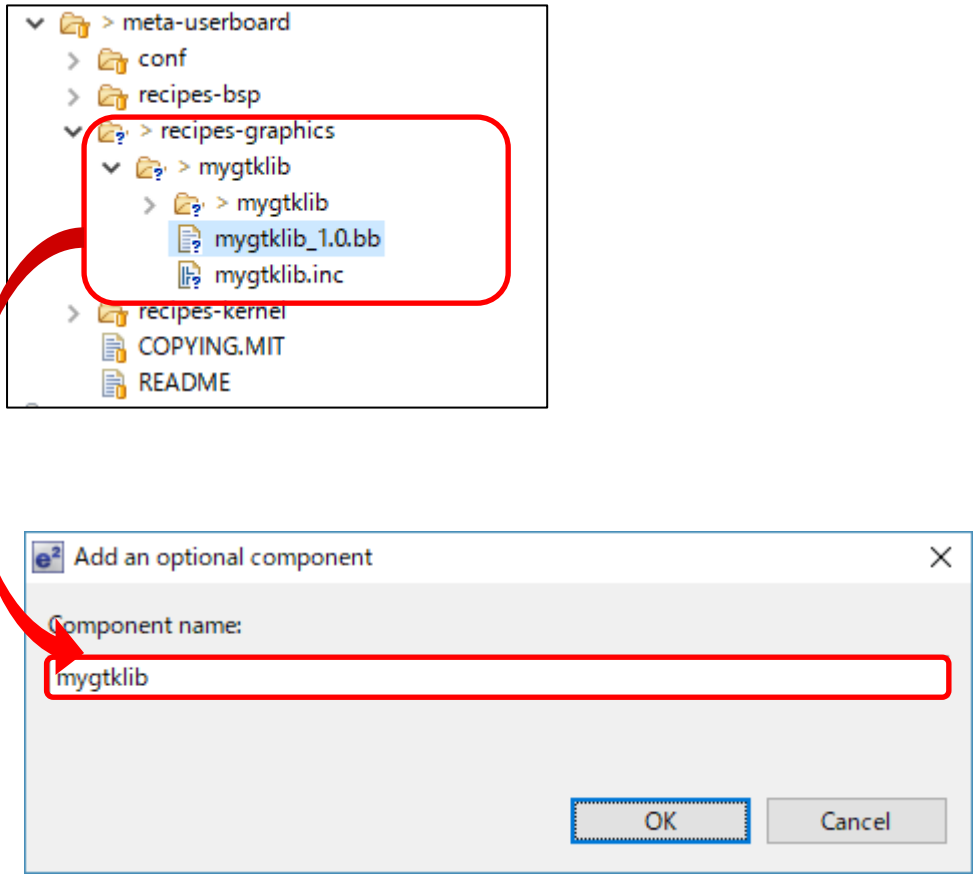


When the drag-and-drop operation in the e^2 studio is not possible, take the following steps to add the recipe.

(1) Right-click on the Yocto recipe file to be added in Windows File Explorer and select [Copy] from the pop-up menu.
(2) Right-click on the folder to which the Yocto recipe is to be added in the [Project Explorer] pane of the e^2 studio and select [Paste] from the pop-up menu.



3. To include the added Yocto recipe as a target for building, open the [Optional components] tabbed page in the window for setting up the build process, click on the [Add] button, enter the name of the added Yocto recipe (the "name" part of the bb filename represented in the form name_version.bb), and click on the [OK] button.
   Any function added as an optional component here will be included in building and incorporated in the image file.



Example: Enter "mygtklib", the "name" part of the bb filename "mygtklib_1.0.bb" as the component name.

4. To build from the added Yocto recipe, open the [BSP] tabbed page in the window for setting up the build process and select [Custom BSP project on the workspace:], and proceed with the build process.

5. On the target board, confirm the operation of the image file which was built in step 3. If there are no problems with the operation, register the added Yocto recipe with the Git repository. For how to register, see 3.14.2 Registering with the Git.

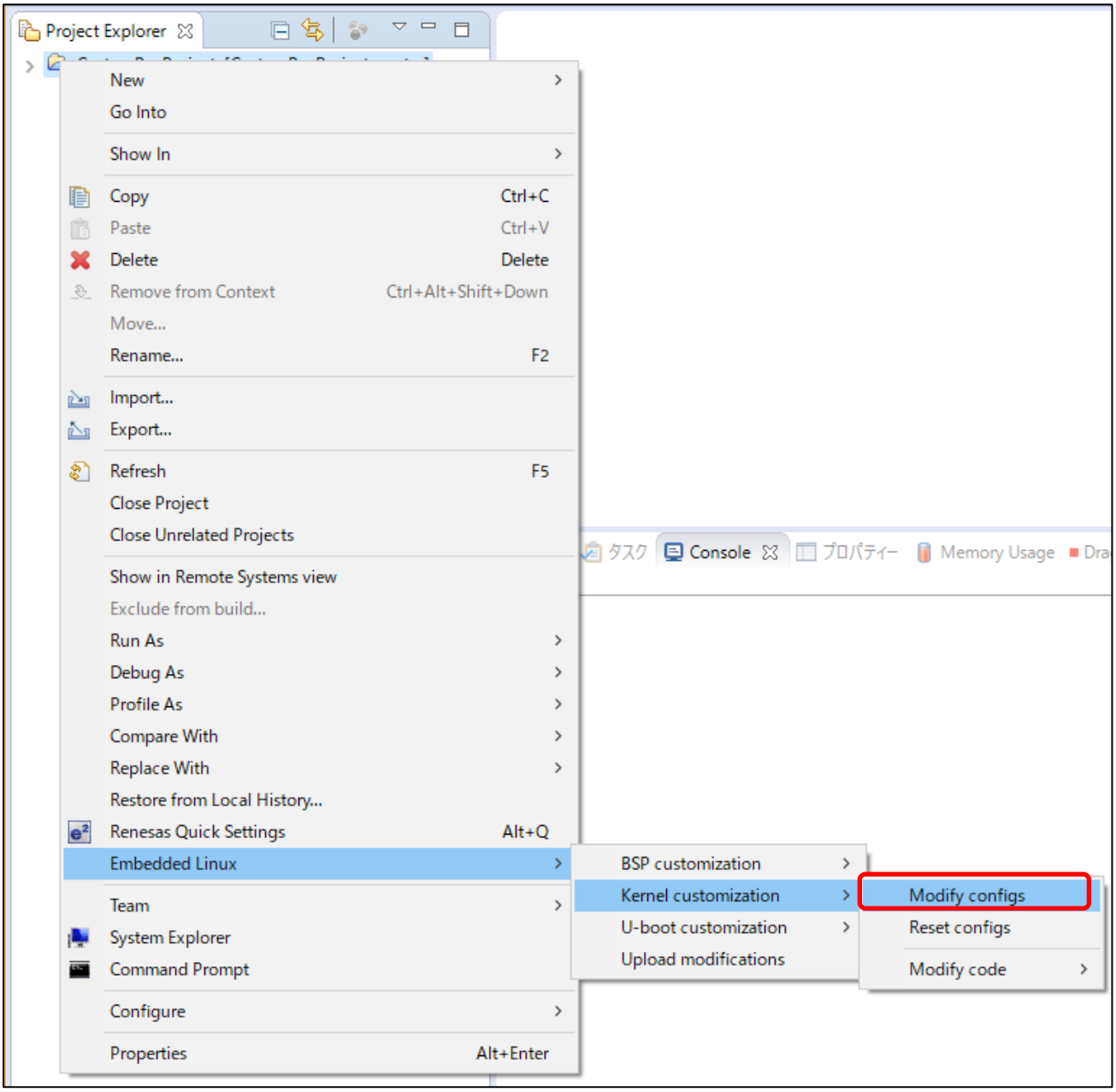
3.12 Customizing the Kernel

This section describes the procedure for customizing the kernel.

3.12.1 Changing the Kernel Configuration

This subsection describes how to change the kernel configuration.

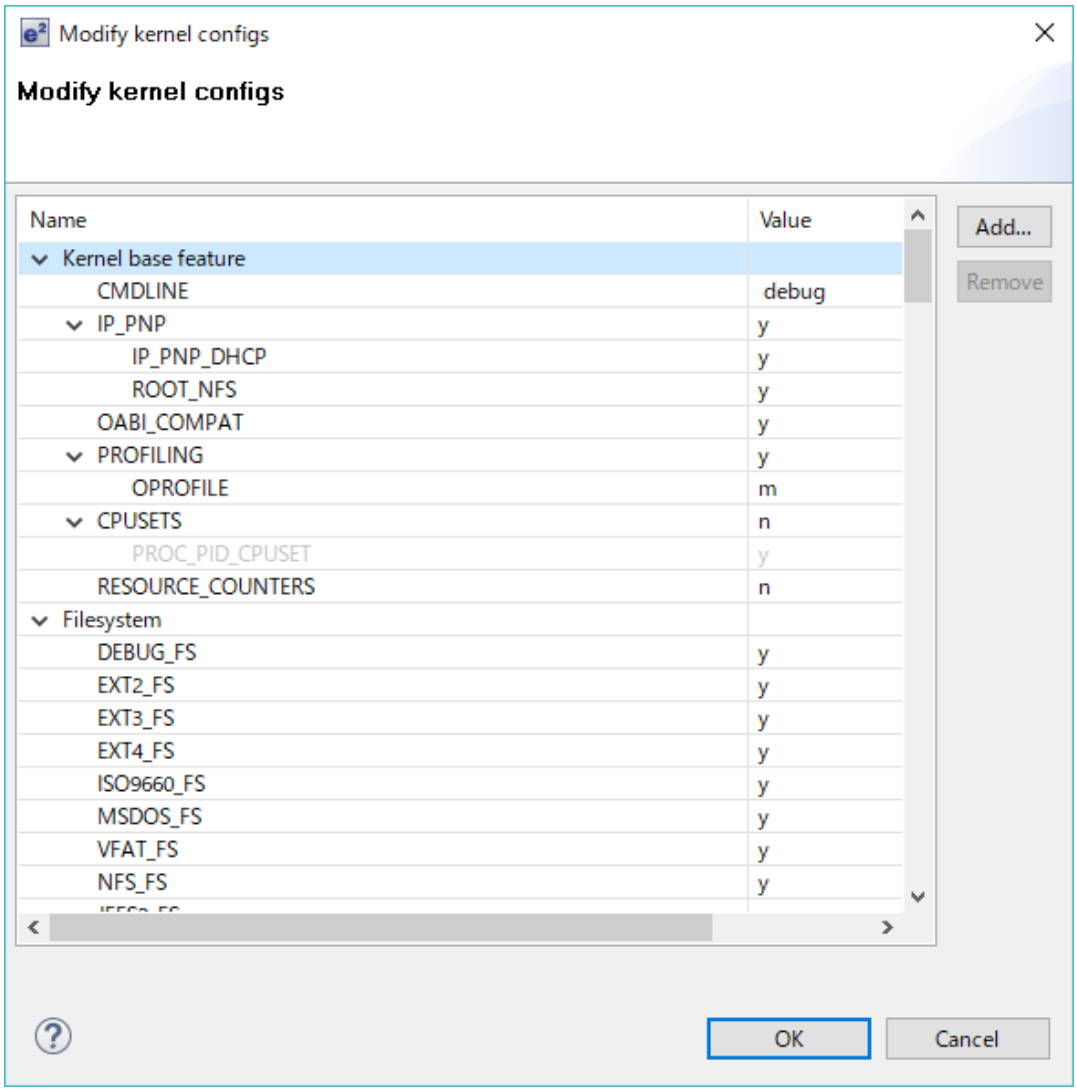
1. Right-click on the embedded Linux custom BSP project in the [Project Explorer] pane and select [Embedded Linux] -> [Kernel customization] -> [Modify configs].



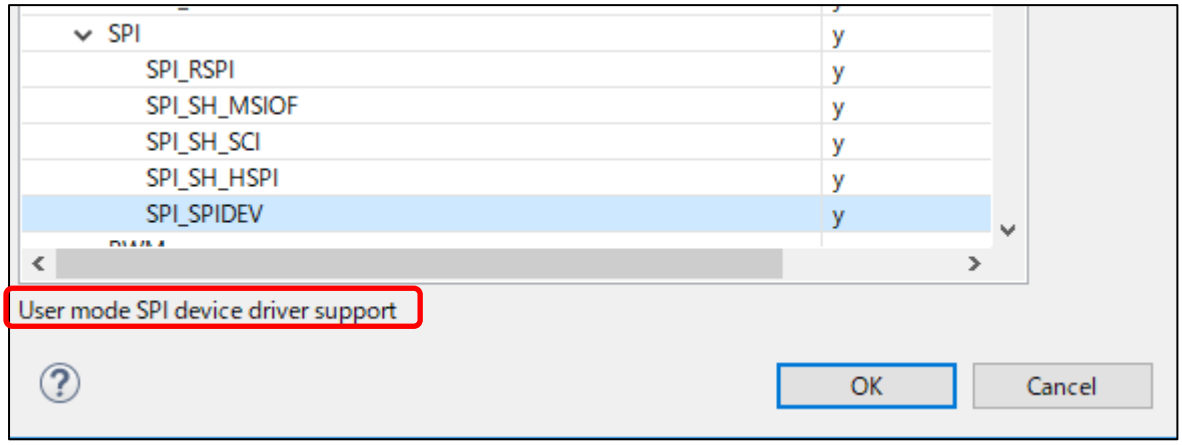
2. The [Modify kernel configs] window opens. Select the [Value] for each function (Name) to be changed from the following values.
   Note: Some items cannot be selected depending on the functions to be changed.

- y: Enables the function.
- m: Incorporates the function as the external module.
- n: Disables the function.

When [Kernel base feature] -> [CMDLINE] is selected, the [Value] is subject to text editing. As the default value, "debug", which specifies the Kernel output message as the DEBUG priority, has been input.



When select each function in the [Modify kernel configs] window, the overview about selected function is displayed in lower part.



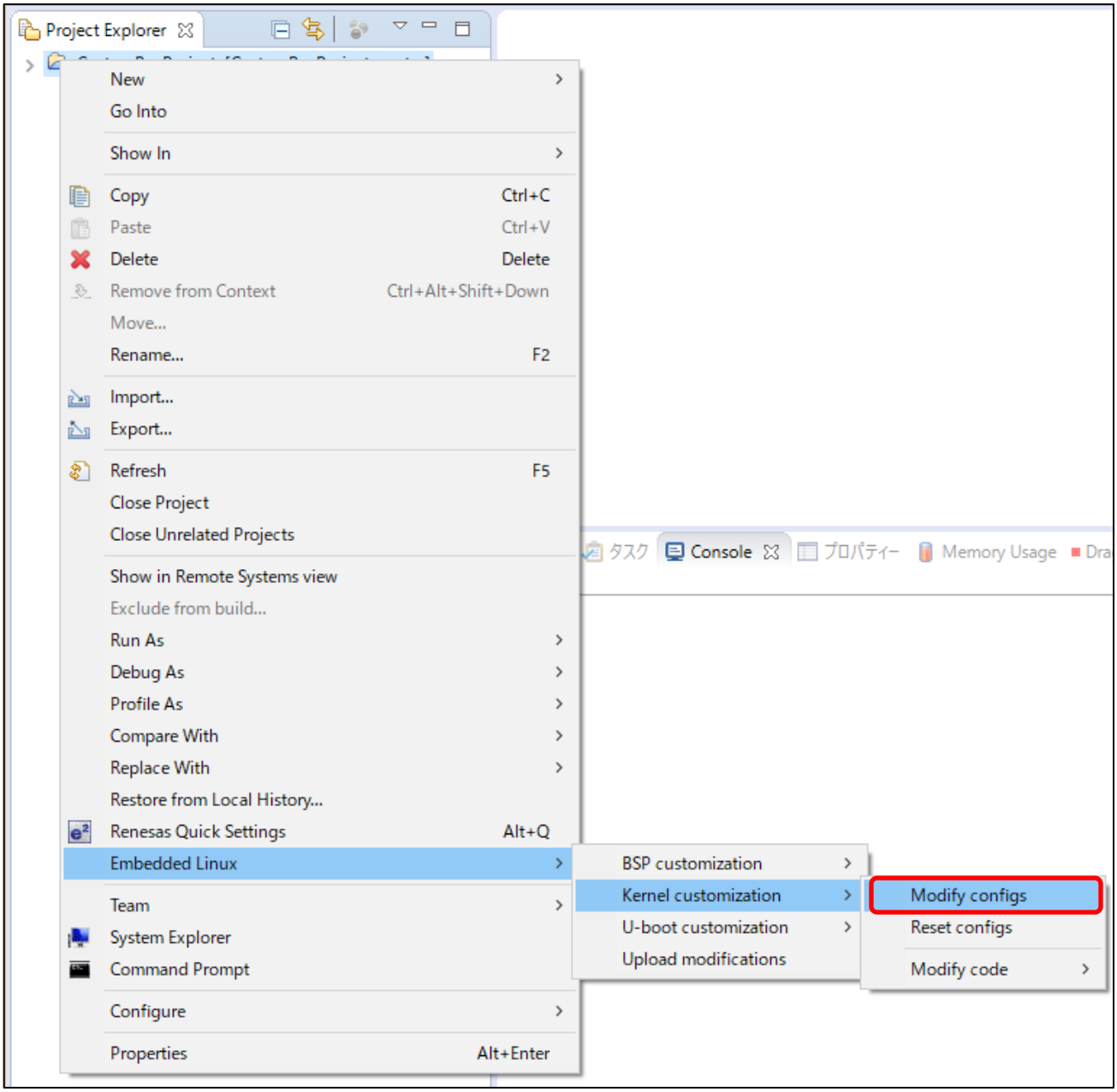
3. To build from the changed kernel configuration, open the [BSP] tabbed page in the window for setting up the build process and select [Custom BSP project on the workspace:], and proceed with the build process.

4. On the target board, confirm the operation of the image file which was built in step 3. If there are no problems with the operation, register the results of changing with the Git repository. For how to register, see 3.14.2 Registering with the Git.

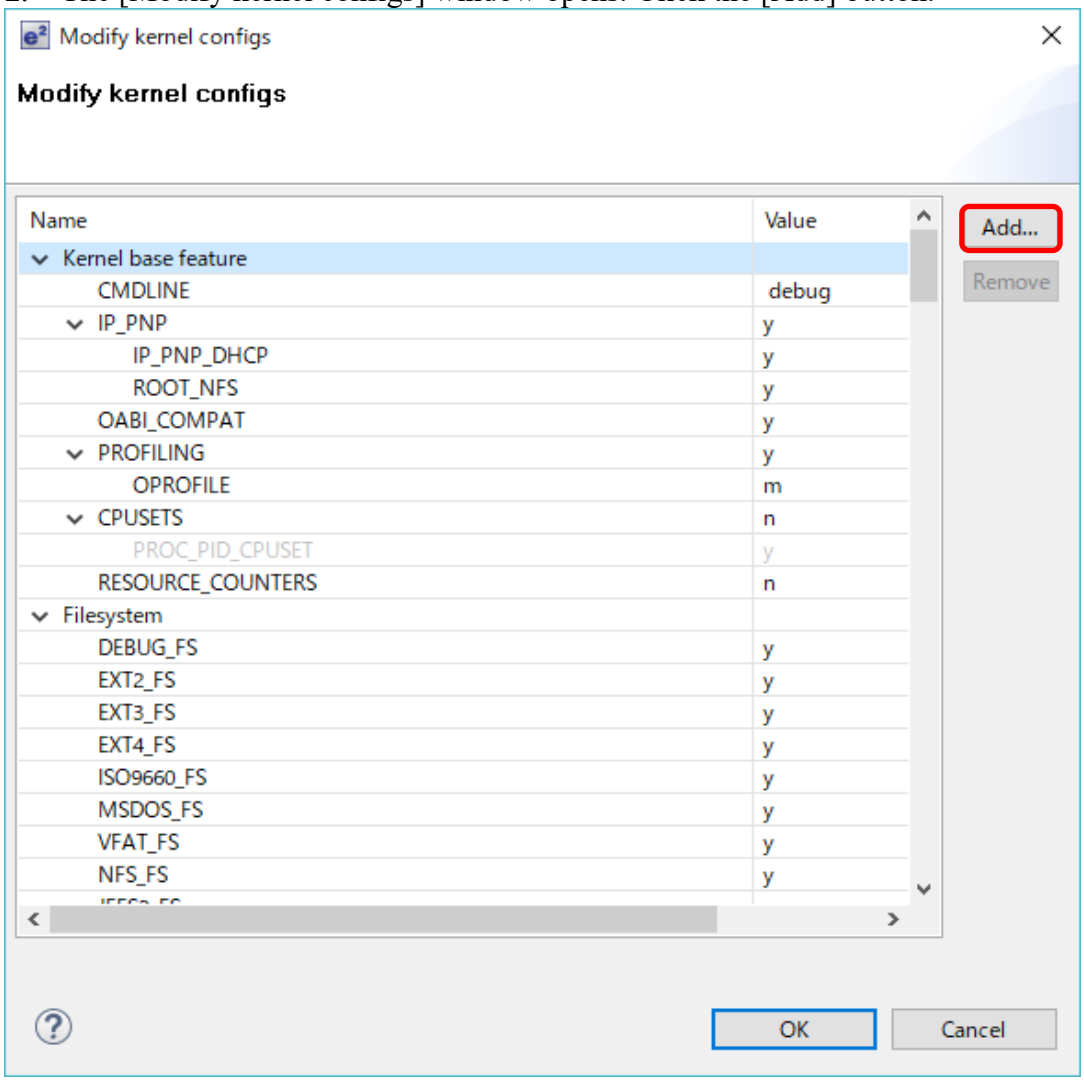
3.12.2 Adding and deleting of the Kernel Configuration function

This subsection describes how to add and delete to the Kernel Configuration.

1. Right-click on the embedded Linux custom BSP project in the [Project Explorer] pane and select [Embedded Linux] -> [Kernel customization] -> [Modify configs].



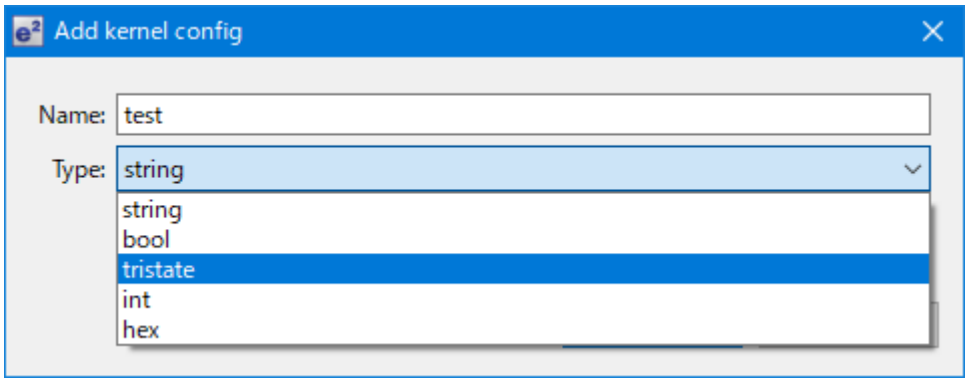
2. The [Modify kernel configs] window opens. Click the [Add] button.



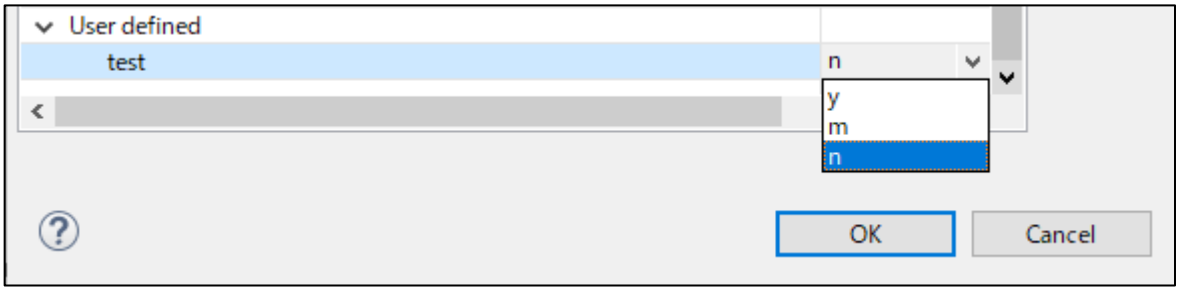
3. The [Add kernel config] window opens. Enter the function name to be added in [Name] and open the [Type] tabbed page. Select the value to specify for it form the following then click the [OK] button.

- string: Enter by string.
- bool : Select from "y" (*1) or "n" (*2)
- tristate: Select from "y"(*1) or "m"(*3) or "n"(*2).
- int :Enter by number. (Decimal)
- hex : Enter by number. (Hexadecimal)

*1 Enables the function.
*2 Disables the function.
*3 Incorporates the function as the external module.



4. The function is added under the [User defined] at the bottom of the [Modify kernel configs] window. Select or enter the [Value].
   * Initial value of "bool" and "tristate" are "n".



5. When building a Kernel that added or deleted a function, open the [BSP] tabbed page in the window for setting up the build process and select [Custom BSP project on the workspace], and execute the build.

6. On the target board, confirm the operation of the image file which was built in step 3. If there are no problems with the operation, register the results of change with the Git repository. For how to register, see 3.14.2 Registering with the Git

[Point for Caution on Adding function to the Kernel Configuration]

The function name add to the Kernel Configuration by this method must use the name in following file. The file changes according to the BSP version.

- BSP ver1.x.x: "MACHINE Configuration" _defconfig
- BSP ver2.x.x: shmobile_defconfig, linux-renesas_4.4.bb

The "xx defconfig" can download by select the directory in "arc/arm/configs" using the full custom mode in editing the Kernel Code. For how to the full custom mode editing in the Kernel Code, see 3.12.4 Editing the Kernel Code. It is described as follows.

```
#
# Automatically generated file; DO NOT EDIT.
# Linux/arm 3.10.31-ltsi Kernel Configuration
#
CONFIG_ARM=y
CONFIG_MIGHT_HAVE_PCI=y
CONFIG_SYS_SUPPORTS_APM_EMULATION=y
CONFIG_HAVE_PROC_CPU=y
CONFIG_NO_IOPORT=y
CONFIG_STACKTRACE_SUPPORT=y
CONFIG_LOCKDEP_SUPPORT=y
CONFIG_TRACE_IRQFLAGS_SUPPORT=y
CONFIG_RWSEM_GENERIC_SPINLOCK=y
CONFIG_ARCH_HAS_CPUFREQ=y
CONFIG_GENERIC_HWEIGHT=y
CONFIG_GENERIC_CALIBRATE_DELAY=y
CONFIG_NEED_DMA_MAP_STATE=y
CONFIG_VECTORS_BASE=0xffff0000
CONFIG_ARM_PATCH_PHYS_VIRT=y
CONFIG_GENERIC_BUG=y
CONFIG_DEFCONFIG_LIST="/lib/modules/$UNAME_RELEASE/.config"
CONFIG_IRQ_WORK=y
CONFIG_BUILDTIME_EXTABLE_SORT=y
```

Each function is described as "CONFIG_<function name>" in the "xx defconfig". The function name add to the Kernel Configuration must set as a string with "CONFIG" removed from "CONFIG_<function name>".

Example) CONFIG_ARM -> "ARM"

The "linux-renesas_4.4.bb" is stored in directory of "/meta-renesas/meta-rzg1/recipes-kernel/linux". It is described as follows.

```
do_configure_append_iwg22m() {
        configure_ravb

        kernel_configure_variable STMPE_I2C y
        kernel_configure_variable MFD_STMPE y
        kernel_configure_variable TOUCHSCREEN_STMPE y
        kernel_configure_variable TOUCHSCREEN_STMPE_CALIBRATION_WORKAROUND y
        kernel_configure_variable SOC_CAMERA_OV772X y
        kernel_configure_variable USB_OHCI_LITTLE_ENDIAN y
        kernel_configure_variable USB_GADGETFS m
        kernel_configure_variable USB_LIBCOMPOSITE m
        kernel_configure_variable USB_MASS_STORAGE m
        kernel_configure_variable USB_F_MASS_STORAGE m
        kernel_configure_variable USB_EHCI_ROOT_HUB_TT y
        kernel_configure_variable USB_XHCI_HCD_DEBUGGING y
        kernel_configure_variable USB_OTG y
        kernel_configure_variable USB_ARCH_HAS_OHCI y
        kernel_configure_variable USB_ARCH_HAS_EHCI y
        kernel_configure_variable USB_ARCH_HAS_XHCI y

        yes '' | oe_runmake oldconfig
}
```

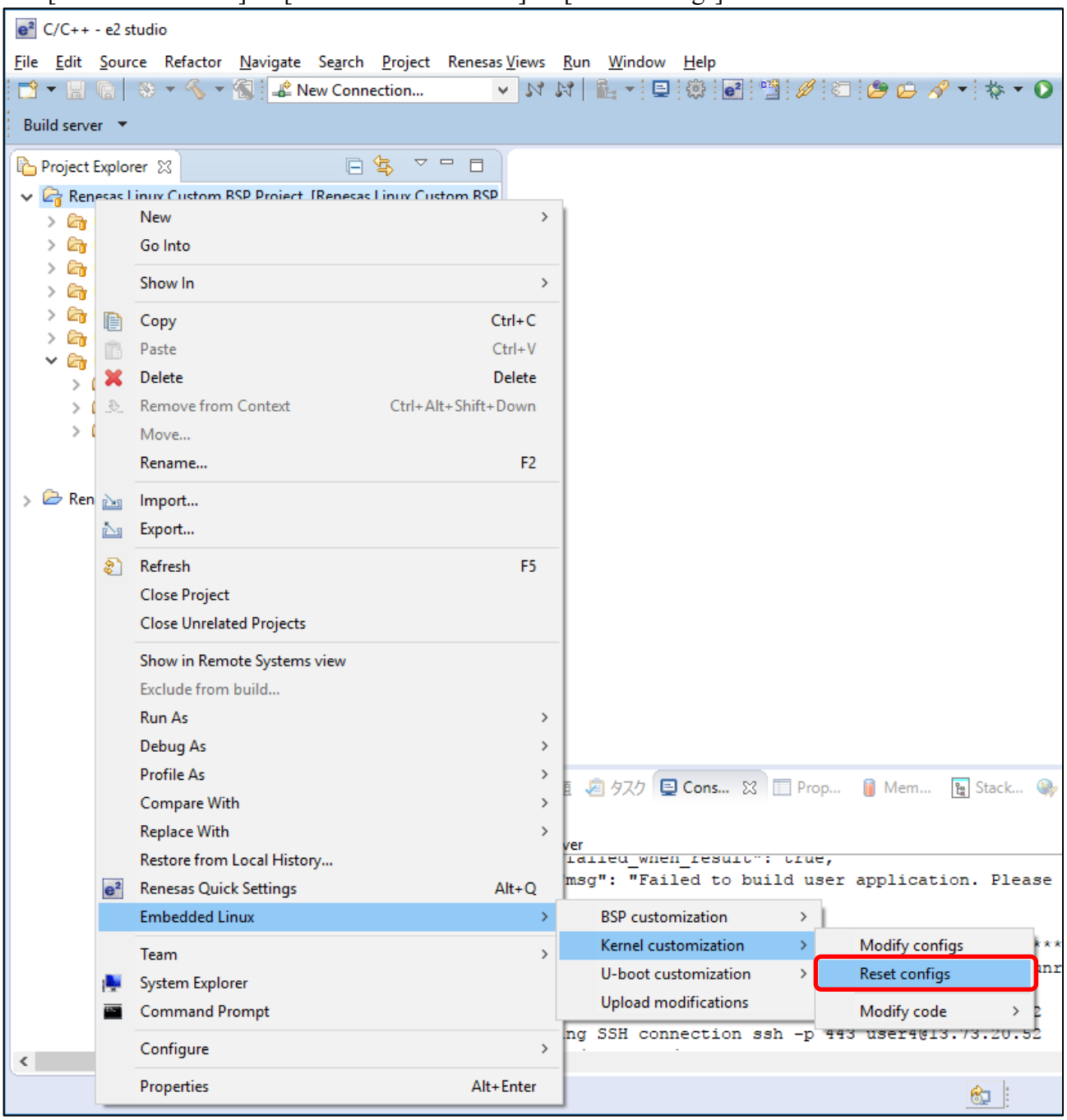
The function of each board is defined the structure named "do_configure_append()" and "do_configure_append_<MACHINE Configuration> ()".
The function name add to the Kernel Configuration must set the name defined in these structures.

Example) STMPE_I2C -> "STMPE_I2C"

3.12.3 Initializing the Kernel Configuration

This subsection describes how to initialize the changed kernel configuration.

1. Right-click on the embedded Linux custom BSP project in the [Project Explorer] pane and select [Embedded Linux] -> [Kernel customization] -> [Reset configs].

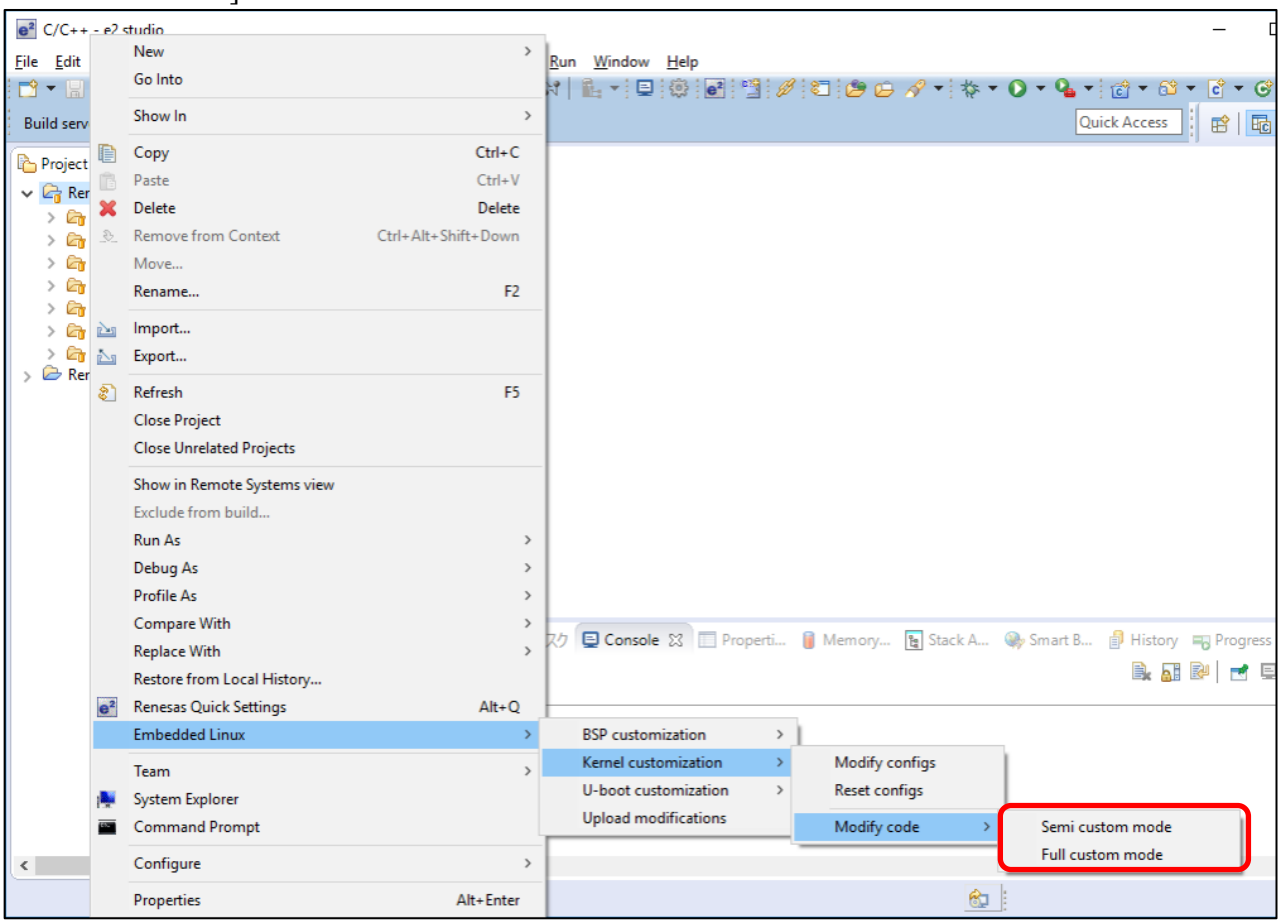


2. To reflect the initialized kernel configuration in a recipe, register the contents of initialization with the Git. For how to register, see 3.14.2 Registering with the Git.

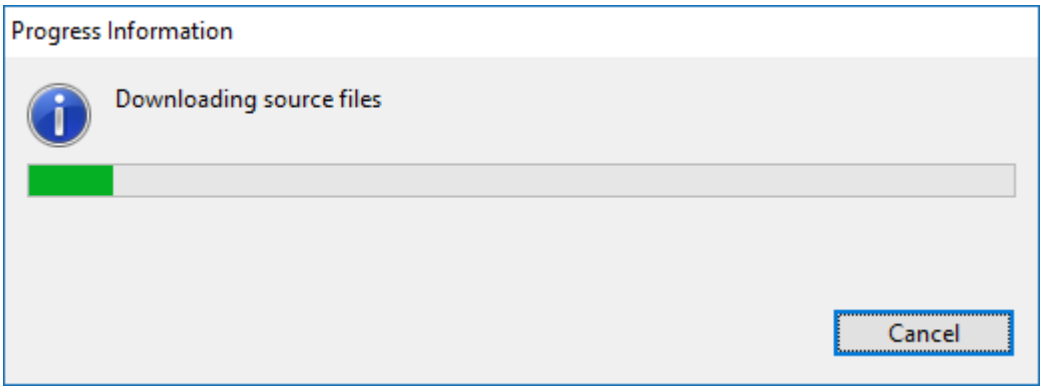
3.12.4 Editing the Kernel Code

This subsection describes how to edit the kernel code.

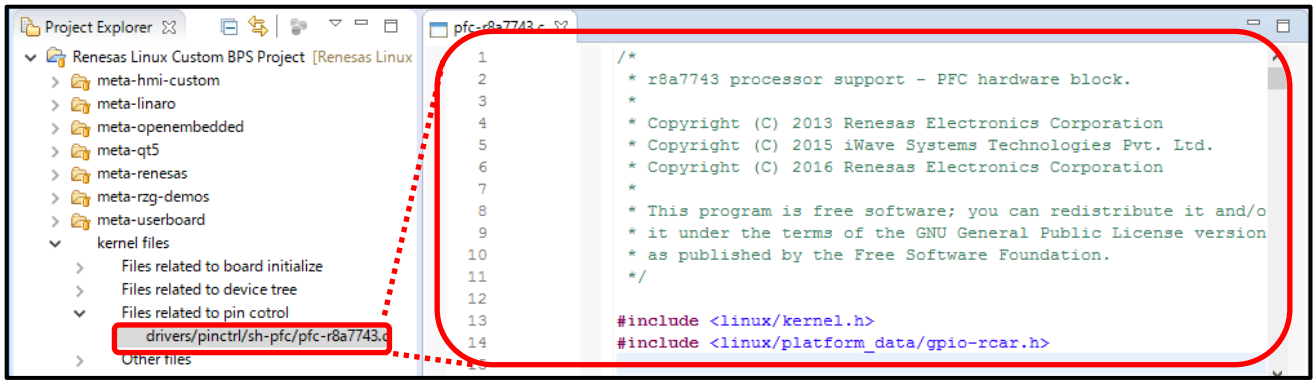
1. Right-click on the embedded Linux custom BSP project in the [Project Explorer] pane and select [Embedded Linux] -> [Kernel customization] -> [Modify code] -> [Semi custom mode] or [Full custom mode].



2. The following window opens. Downloading of the kernel code from the build server to the client PC starts.



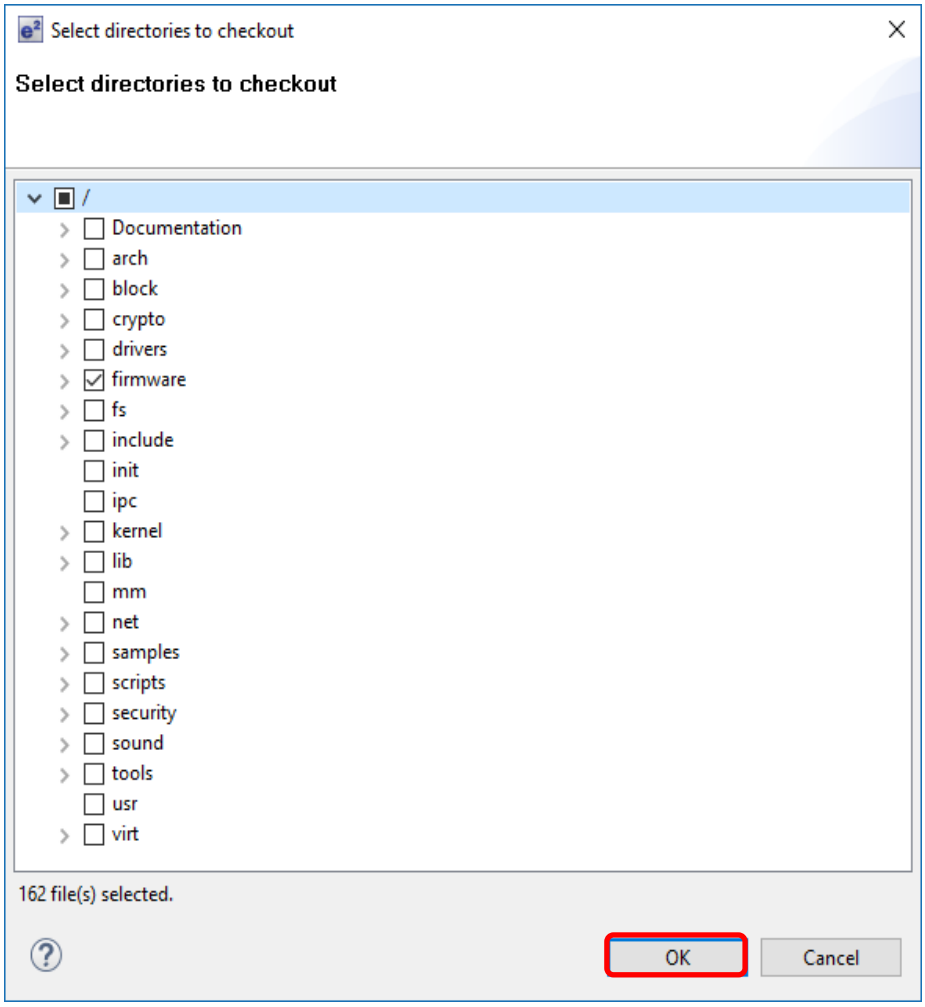
3. Selecting the mode
   (1) Selecting the semi custom mode
   On completion of downloading, [kernel files] is created under the embedded Linux custom BSP project in the [Project Explorer] pane.
   Opening [kernel files] allows you to see the kernel files classified in four categories.
   Double-click on or drag and drop the file to be modified to the editor window of the e^2 studio screen to open and edit the file.



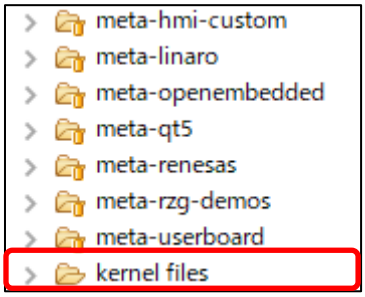
Category	Description
Files related to pin control	A set of files for changing pin settings on the target board.
Files related to device tree	A set of files for changing the device tree in which hardware information of the target board is described.
Files related to initialization of the board	A set of files for changing system-related information such as memory size of the target board.
Other files	A set of files for changing methods of power management and memory management of the target board.

(2) Selecting the full custom mode

On completion of downloading, the dialog box for selecting directories or files to be edited opens. Check the names of directories or files to be edited and click on the [OK] button.

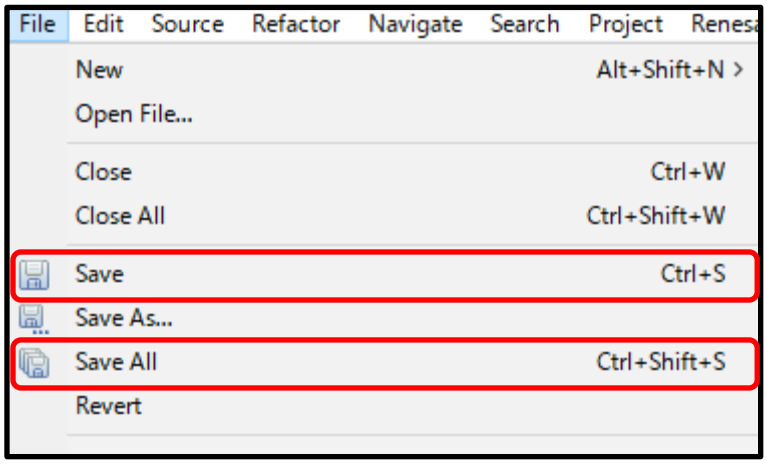


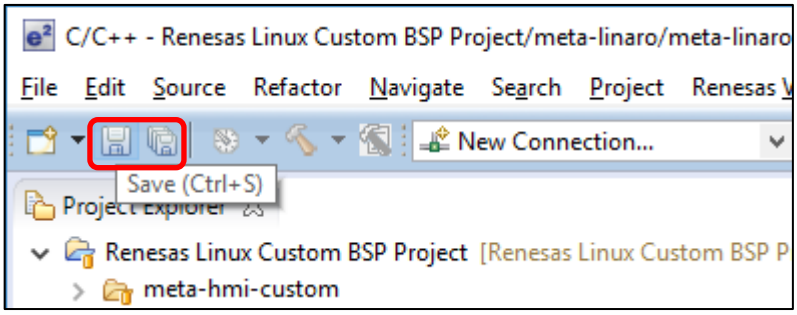
The [kernel files] will then be created under the embedded Linux custom BSP project in the [Project Explorer] pane, and the selected directories or files will be checked out.



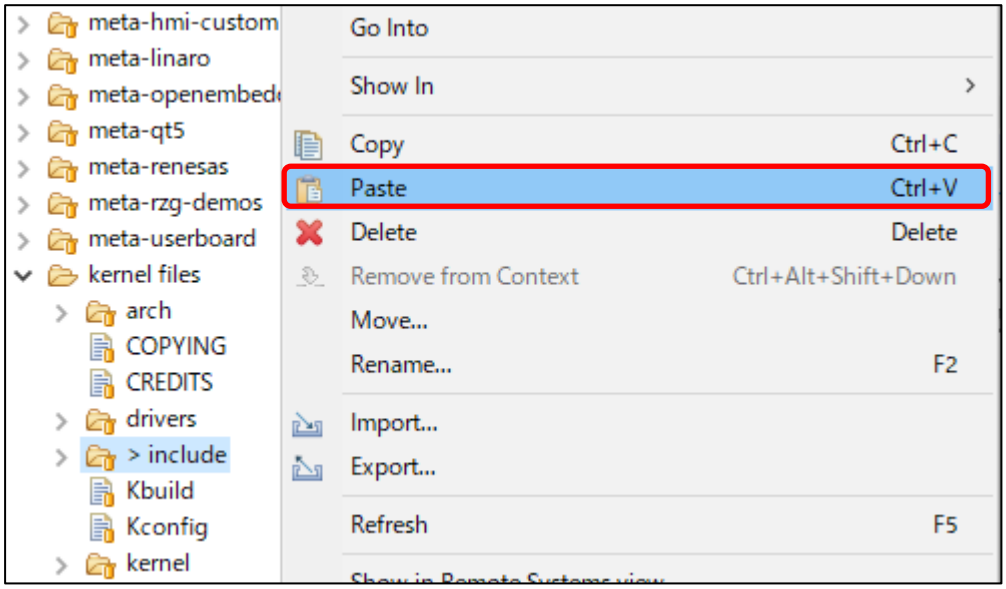
Opening [kernel files] allows you to see these files. Double-click on or drag and drop the file to be modified to the editor window of the e^2 studio screen to open and edit the file.

4. Select [Save] or [Save All] from [File] of the menu bar at the top of the window or from the icons at the top of the window to save the contents of editing.

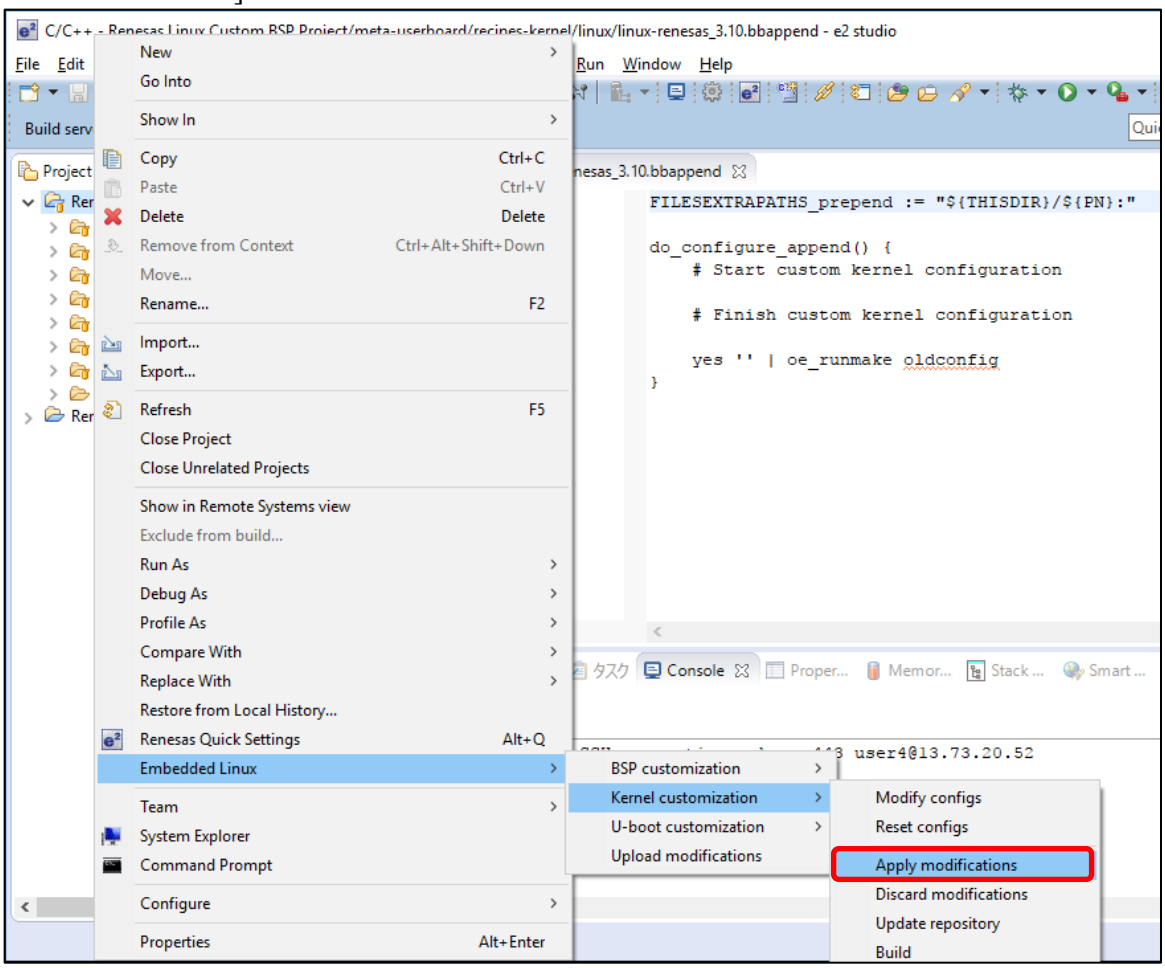




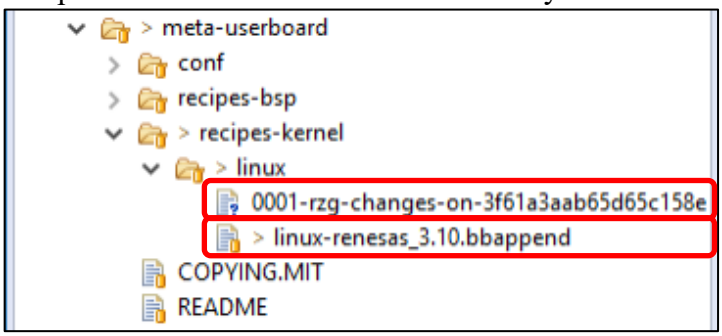
5. In case of adding a new kernel file to the directory instead of editing an existing kernel file, copy the new kernel file, select the directory you want to add, and then right-click and paste.



6. When editing is completed, right-click on the embedded Linux custom BSP project in the [Project Explorer] pane and select [Embedded Linux] -> [Kernel customization] -> [Apply modifications].



7. Selecting [Apply modifications] automatically creates the Yocto recipe patch according to the contents of modification. In addition, the descriptions for applying the automatically created patch to the BSP are also automatically written to the recipe.



8. To build from the results of editing, open the [BSP] tabbed page in the window for setting up the build process and select [Custom BSP project on the workspace], and proceed with the build process.

9. On the target board, confirm the operation of the image file which was built in step 7. If there are no problems with the operation, select [Embedded Linux] -> [Kernel customization] -> [Update repository] to check whether the results of editing by other users do not conflict with yours (conflict between the results of editing).

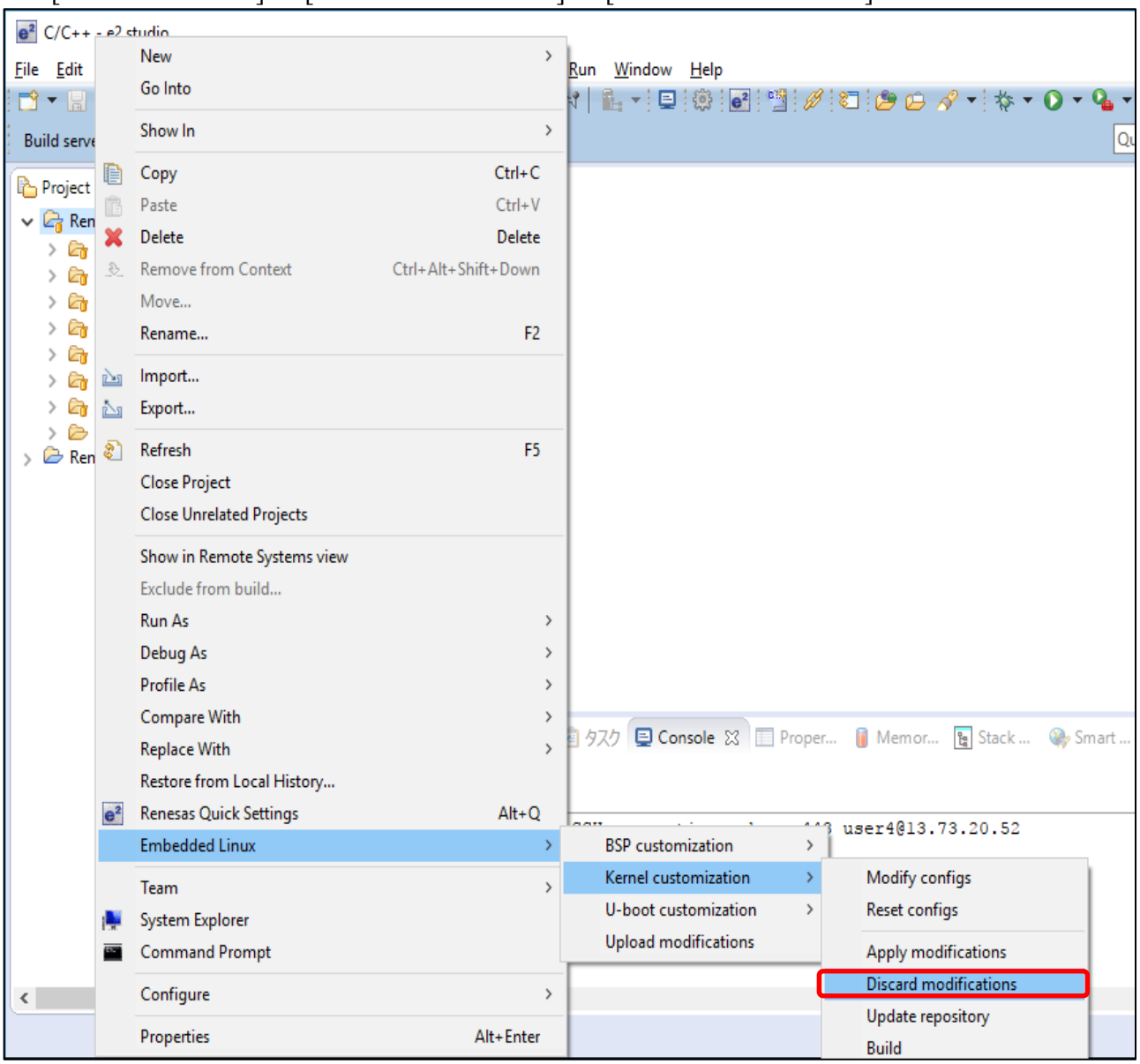
10. If you found a conflict in step 8, check the results of editing at the location where the conflict arose and resolve the conflict. After resolving the conflict, delete the patch file and the description written to the recipe created in response to selecting [Apply modifications] in step 5, and repeat the above steps from step 5.

11. After confirming that operation is correct and that there are no conflicts, register the results of editing with the Git repository. For how to register, see 3.14.2 Registering with the Git.

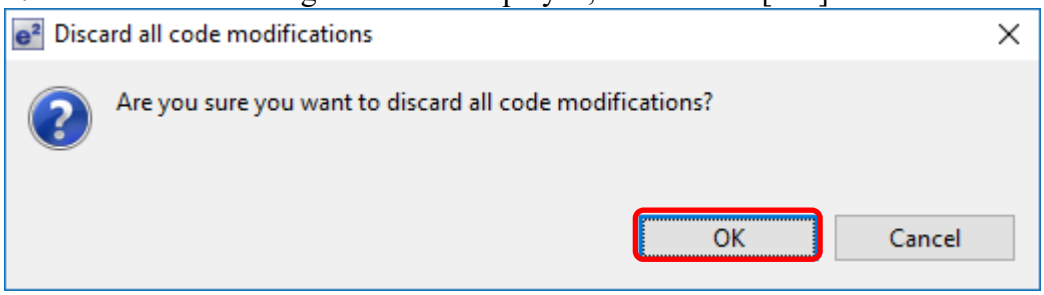
3.12.5 Removing the Contents of Editing the Kernel Code

This subsection describes how to remove the contents of editing the kernel code.

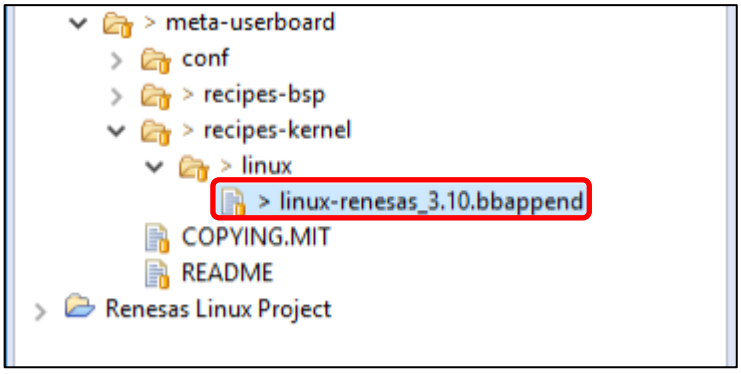
1. Right-click on the embedded Linux custom BSP project in the [Project Explorer] pane and select [Embedded Linux] -> [Kernel customization] -> [Discard modifications].



2. When the following window is displayed, click on the [OK] button.



3. [kernel files] is removed from the embedded Linux custom BSP project in the [Project Explorer] pane.
   In addition, the Yocto recipe patch that has been automatically added during editing is also automatically removed and the code for applying the removal to the BSP is also automatically described.



4. To include the contents of removal as the build target, register the contents of editing with the Git. For how to register, see 3.14.2 Registering with the Git.
   To edit the kernel code again, download the kernel code again as described in 3.12.3 Editing the Kernel Code.

3.12.6 Debugging the Kernel

The Linux kernel image (vmlinux) to be debugged should be downloaded, and then loaded and debugged with the ICE.

Before debugging, the kernel source code should be downloaded and the folder that stores the source code should be specified as the source path in the debugger of the ICE, which enables debugging with reference to the source code.

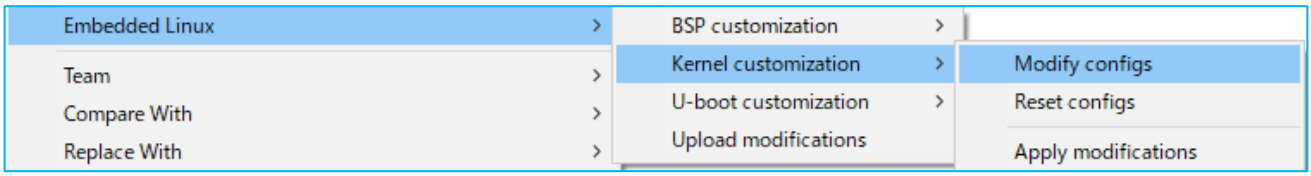
Prepare for the debugging of the kernel with the following procedure, and then debug the kernel with the ICE.

1. Create a Linux kernel image to be debugged
2. Download the Linux kernel image to be debugged

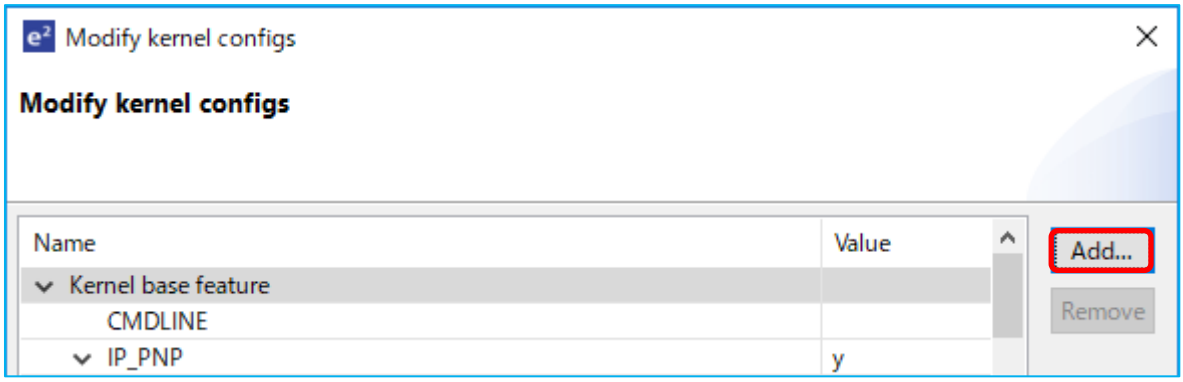
3.12.6.1 Creating a Linux Kernel Image to be Debugged

Create a Linux kernel image to be debugged with the following procedure.

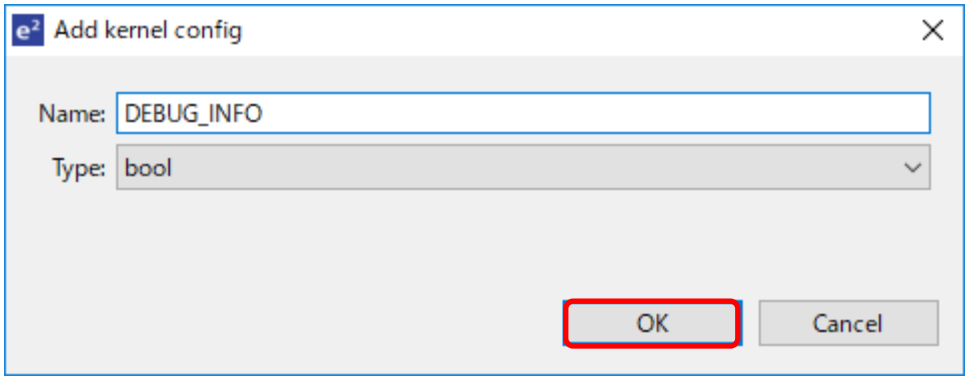
1. Right-click on the embedded Linux custom BSP project and select [Embedded Linux] -> [Kernel customization] -> [Modify configs].



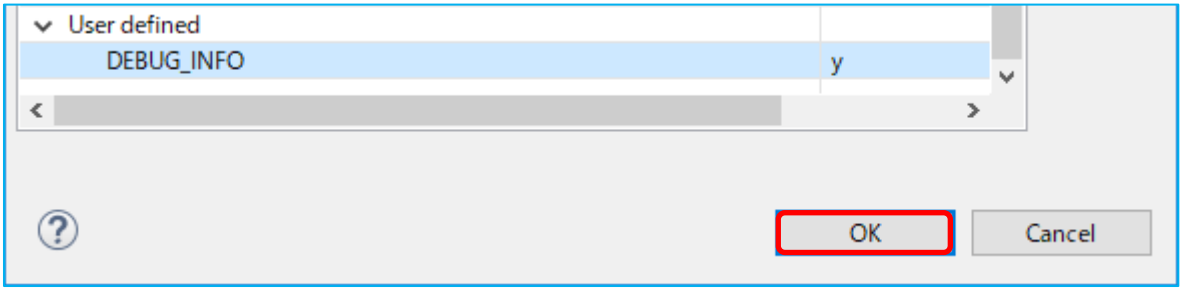
2. Click on the [Add] button in the top-right section of the [Modify kernel configs] window to open the [Add kernel config] window.



3. Enter "DEBUG_INFO" for [Name], select "bool" for [Type], and click on [OK].



4. Change the value for the added "DEBUG_INFO" from "n" to "y" and click on [OK].



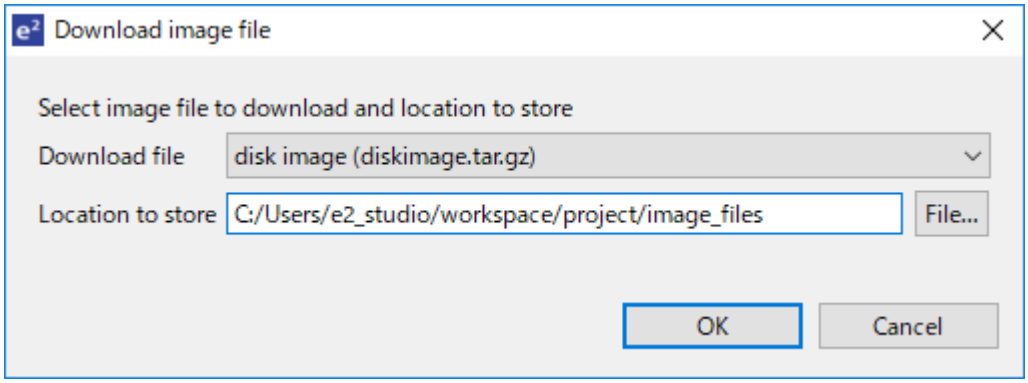
5. Right-click on the embedded Linux custom BSP project, select [Embedded Linux] -> [Kernel customization] -> [Build], specify this embedded Linux custom BSP project for [Custom BSP project on the workspace], and proceed with the build process.

A Linux kernel image to be debugged will be available for download.

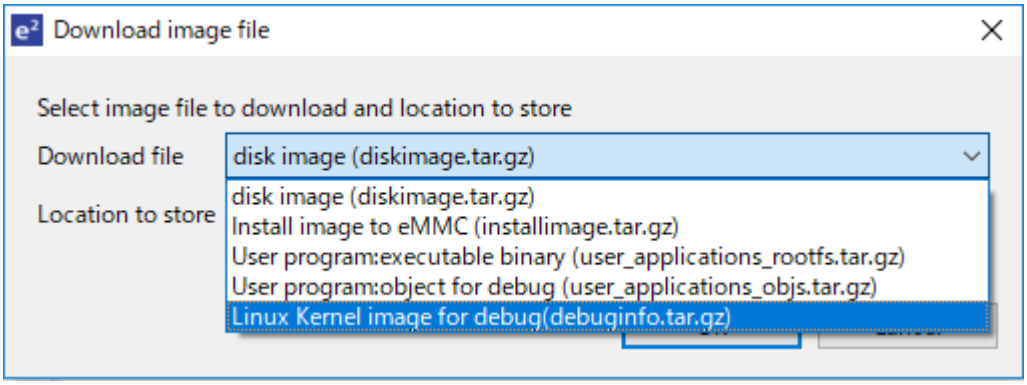
3.12.6.2 Downloading the Linux Kernel Image to be Debugged

Download the Linux kernel image to be debugged with the following procedure.

1. Right-click on the embedded Linux project and select [Embedded Linux] -> [Download] -> [Built image] to open the [Download image file] window.
   Alternatively, this window will automatically be opened when the build process is completed.



2. From the pull-down menu for [Download file], select [Linux Kernel image for debug(debuginfo.tar.gz)] and click on [OK]. The debuginfo.tar.gz file will be downloaded to the folder specified in [Location to store].



3.12.6.3 How to Use the Linux Kernel for Debugging

Decompress the debuginfo.tar.gz file and the following two files will be extracted.

Name	Date modified	Type	Size
vmlinux	9/26/2018 6:12 PM	File	120,854 KB
kernel_build_path.txt	9/26/2018 6:12 PM	Text Document	1 KB

The vmlinux file is the target Linux kernel image to be debugged. Load this file with the ICE and the kernel can be debugged with the debugger in the ICE.

The kernel_build_path.txt file contains the source path to the folder that stores the source files that were used for building the vmlinux file.

Example of the source path in the vmlinux file:
/home/username/RZG1E/iWave/2.0.3/bsp/build/tmp/work-shared/iwg22m/kernel-source/

The vmlinux file contains the source path of the build system where the kernel was built. If the vmlinux file is loaded in its current state, the debugger will attempt to refer to the source path of the build system and be unable to locate the correct source code.

To correctly refer to the source code, the source path in the vmlinux file must be converted to the path to the folder in the client PC where the downloaded kernel source code is stored.

Example of the path to the folder where the downloaded kernel source code is stored:
C:/Users/e2_studio/workspace/CustomBspProject/.rzg_local/kernel/

If the debugger has a function for converting the source path, specify the source path written in the kernel_build_path.txt file as the old path before conversion.
Specify the folder where the downloaded kernel source code is stored as the new source path to be used after conversion, and the debugger can refer to the correct source code for debugging.

If the debugger does not have a function for converting the source path, create folders in the C drive of the client PC so that the folder structure exactly matches the source path written in the kernel_build_path.txt file, and store the kernel source code there.

After this preparation, the kernel can be debugged with the ICE.

The methods for loading the vmlinux file and converting or specifying the source path differ depending on the ICE and its debugger. For these methods, consult with the manufacturer or vendor of the ICE.

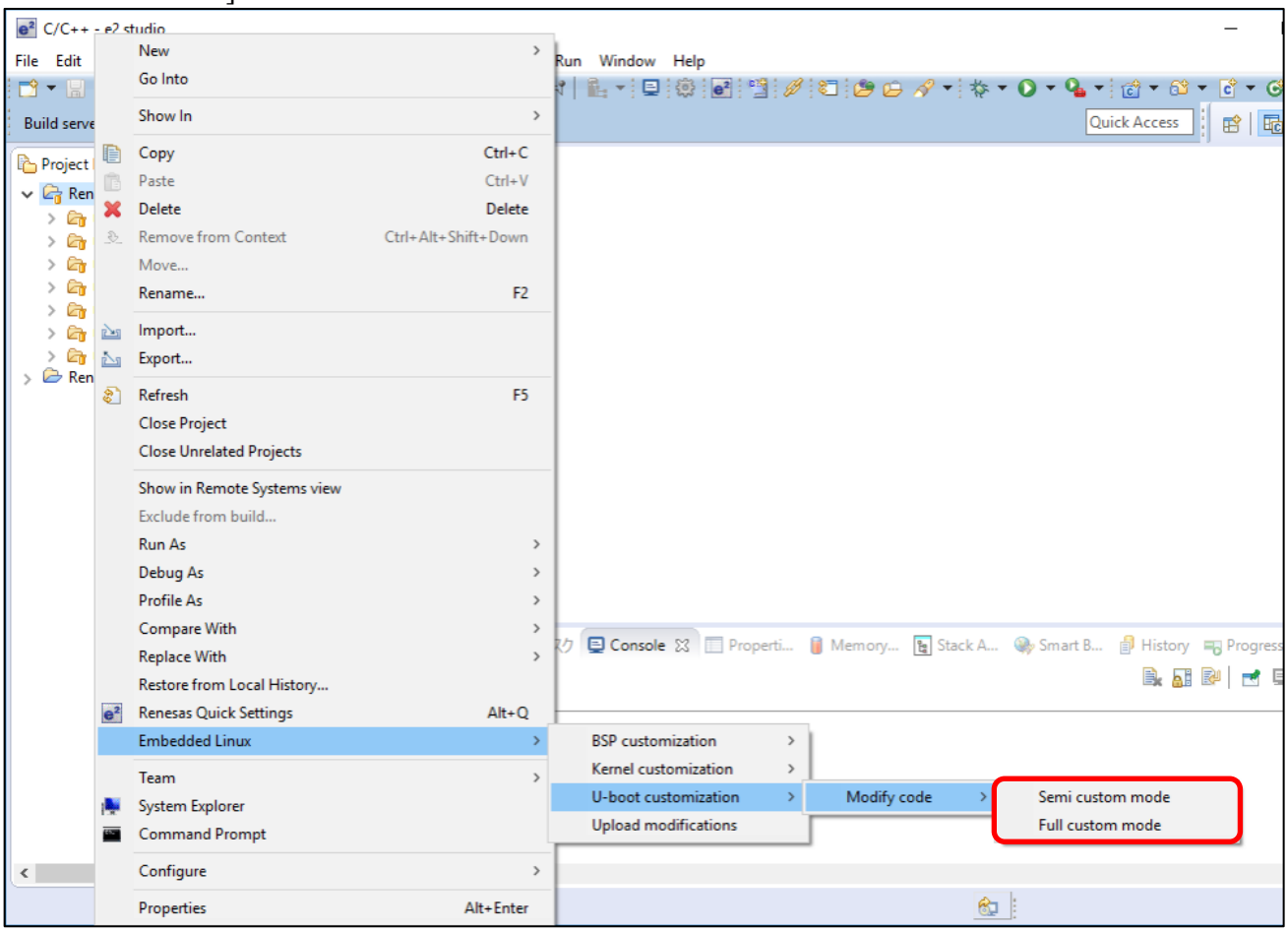
3.13 Customizing U-Boot

This section describes the procedure for customizing U-Boot.

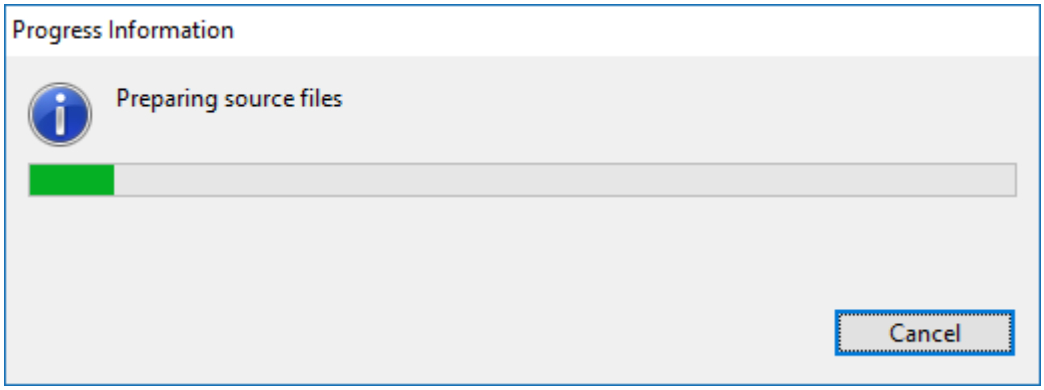
3.13.1 Editing the U-Boot Code

This subsection describes how to edit the U-Boot code.

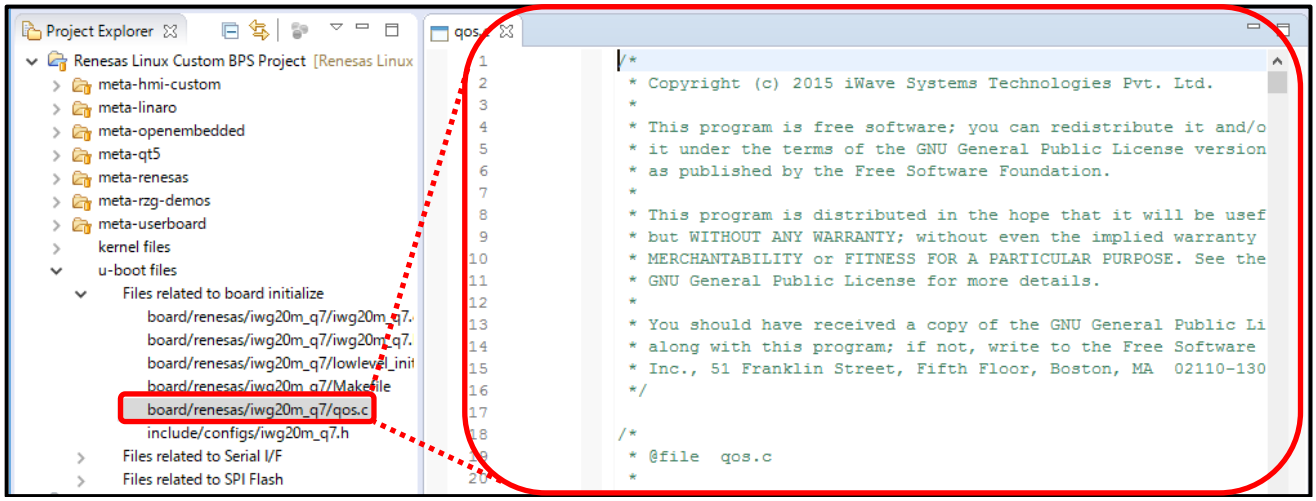
1. Right-click on the embedded Linux custom BSP project in the [Project Explorer] pane and select [Embedded Linux] -> [U-boot customization] -> [Modify code] -> [Semi custom mode] or [Full custom mode].



2. The following window opens. Downloading of the U-Boot code from the build server to the client PC starts.



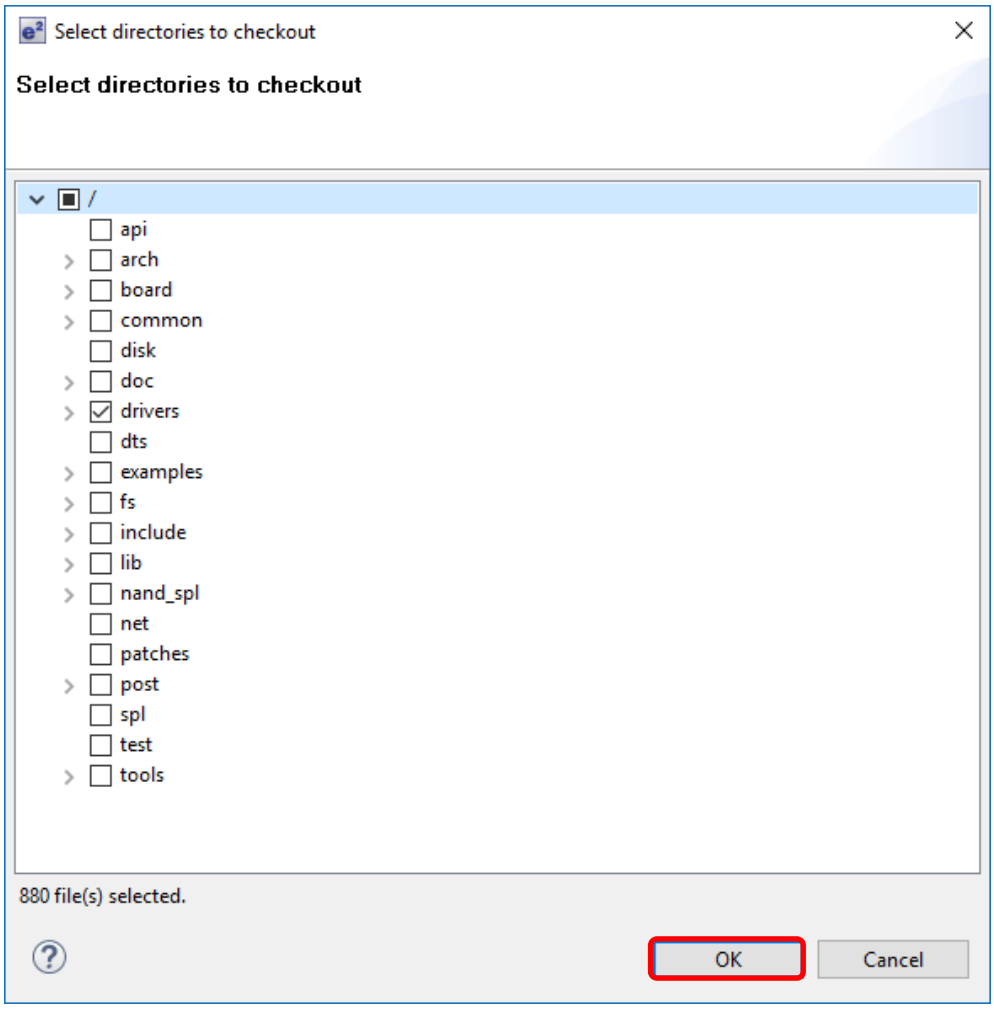
3. Selecting the mode
   (1) Selecting the semi custom mode
   On completion of downloading, [u-boot files] is created under the embedded Linux custom BSP project in the [Project Explorer] pane.
   Opening [u-boot files] allows you to see the U-Boot files classified in three categories.
   Double-click on or drag and drop the file to be modified to the editor window of the e^2 studio screen to open and edit the file.



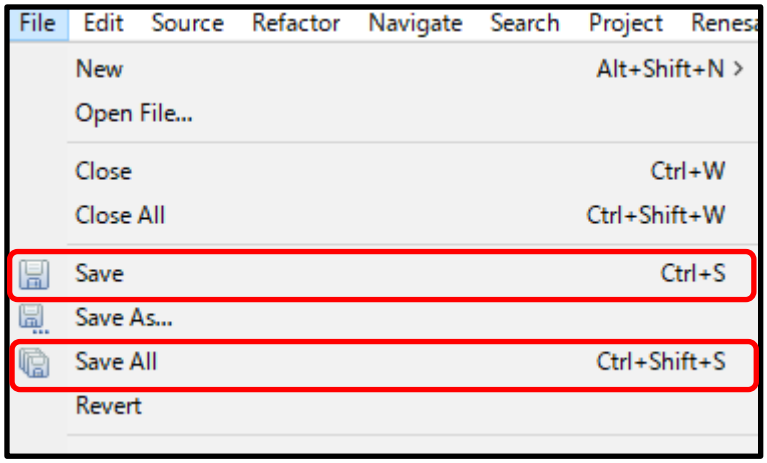
(2) Selecting the full custom mode

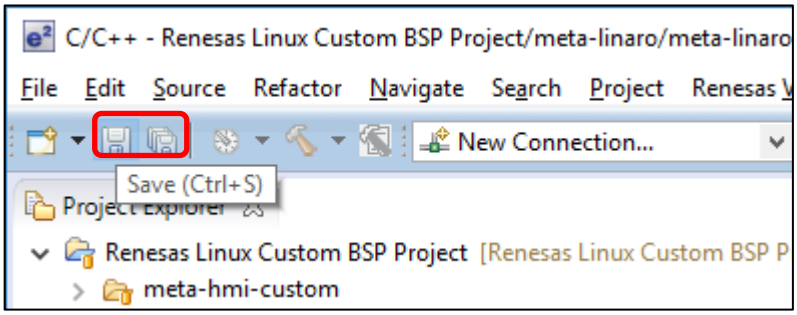
On completion of downloading, the dialog box for selecting directories or files to be edited opens. Check the names of directories or files to be edited and click on the [OK] button. The [u-boot files] will then be created under the embedded Linux custom BSP project in the [Project Explorer] pane, and the selected directories or files will be checked out.

Opening [u-boot files] allows you to see these files. Double-click on or drag and drop the file to be modified to the editor window of the e^2 studio screen to open and edit the file.

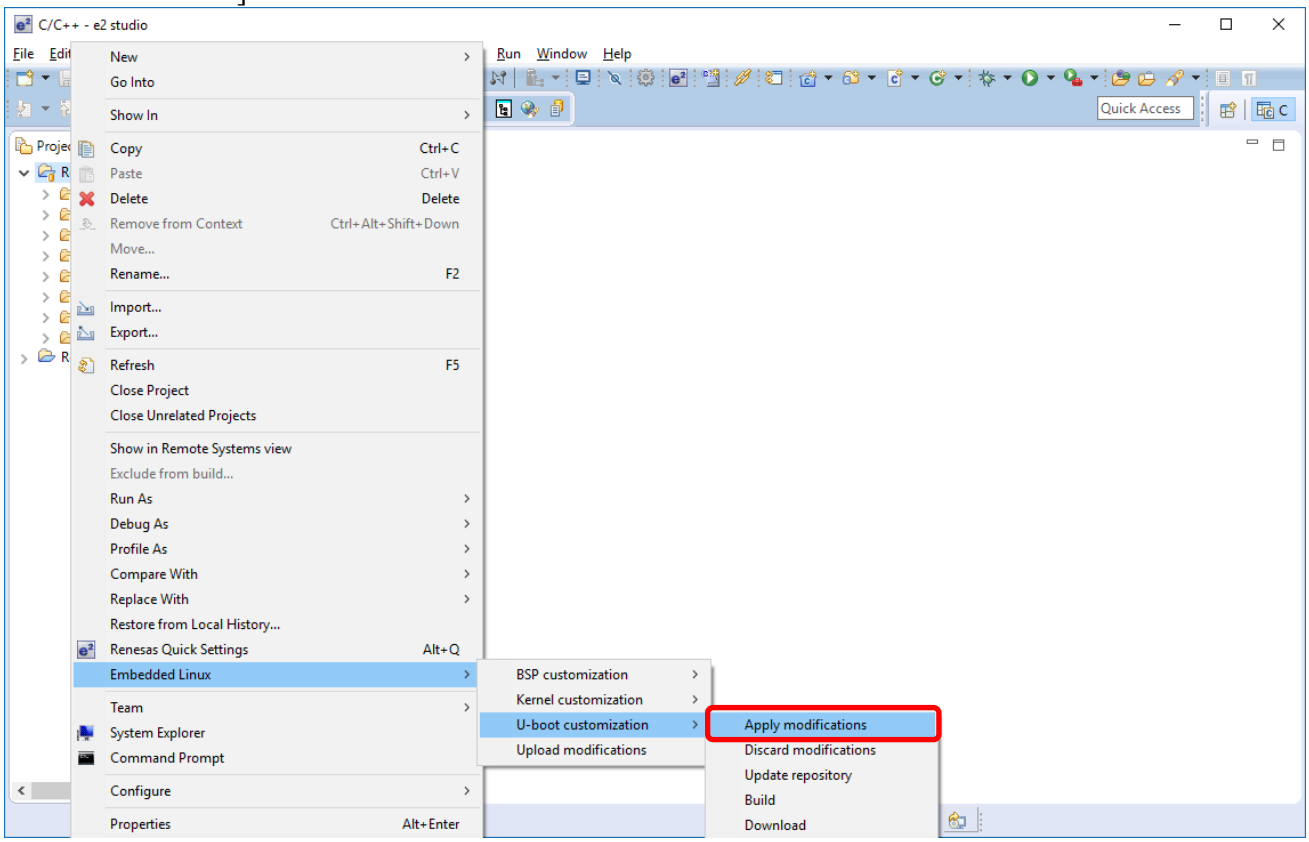


4. Select [Save] or [Save All] from [File] of the menu bar at the top of the window or from the icons at the top of the window to save the contents of editing.

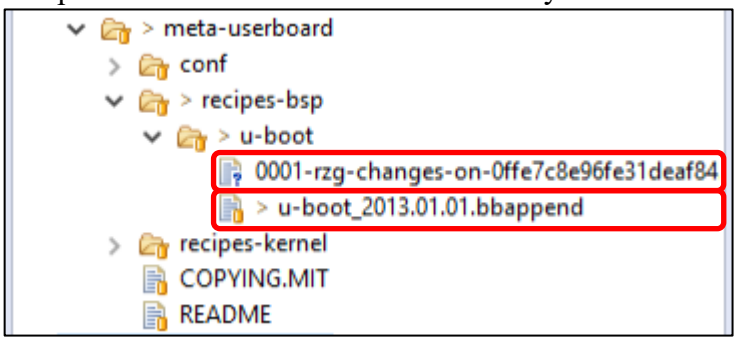




5. When editing is completed, right-click on the embedded Linux custom BSP project in the [Project Explorer] pane and select [Embedded Linux] -> [U-boot customization] -> [Apply modifications].



6. Selecting [Apply modifications] automatically creates the Yocto recipe patch according to the contents of modification. In addition, the descriptions for applying the automatically created patch to the BSP are also automatically written to the recipe.



7. To build from the results of editing, open the [BSP] tabbed page in the window for setting up the build process and select [Custom BSP project on the workspace], and proceed with the build process.

8. On the target board, confirm the operation of the image file which was built in step 7. If there are no problems with the operation, select [Embedded Linux] -> [Kernel customization] -> [Update repository] to check whether the results of editing by other users do not conflict with yours (conflict between the results of editing).

9. If you found a conflict in step 8, check the results of editing at the location where the conflict arose and resolve the conflict. After resolving the conflict, delete the patch file and the description written to the recipe created in response to selecting [Apply modifications] in step 5, and repeat the above steps from step 5.

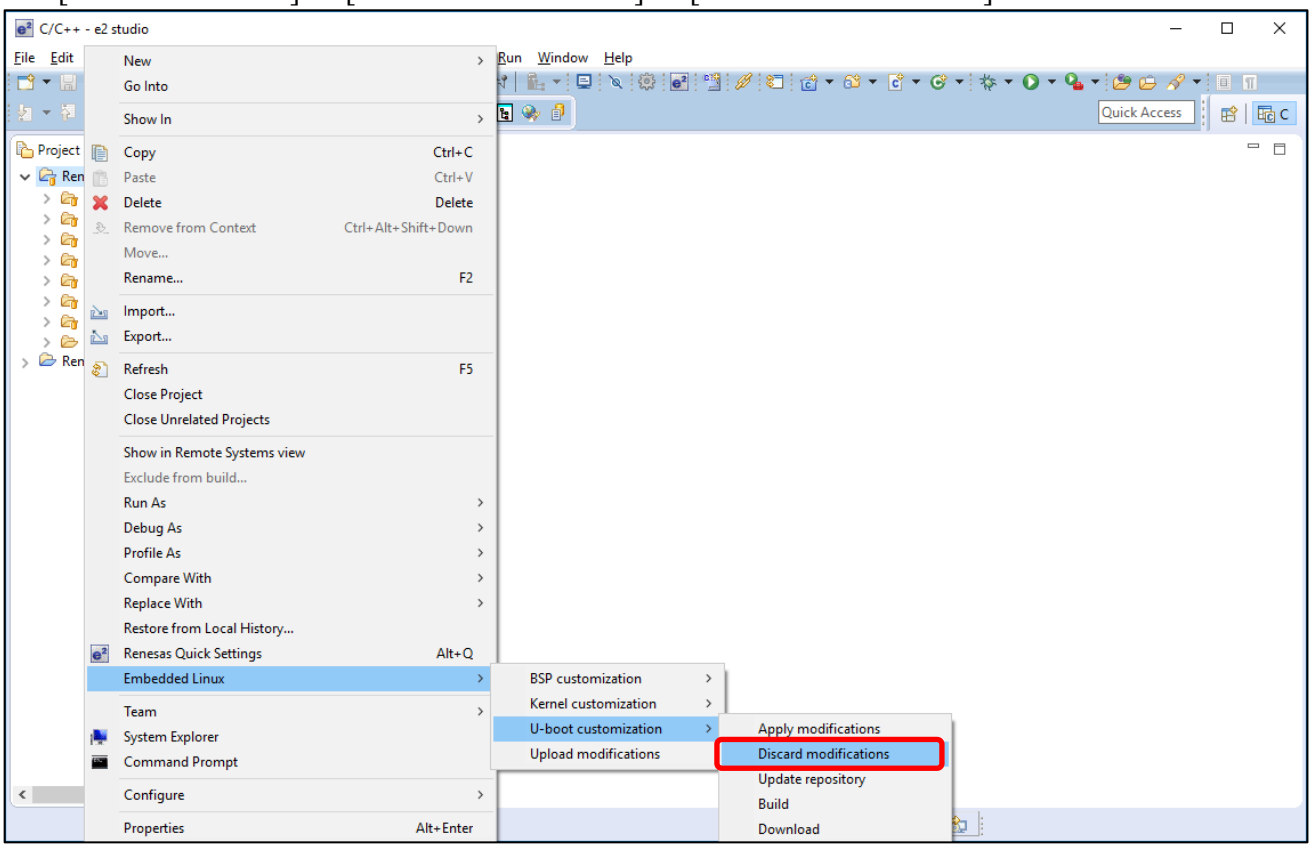
10. After confirming that operation is correct and that there are no conflicts, register the results of editing with the Git repository. For how to register, see 3.14.2 Registering with the Git.

Note: This tool does not support writing of the U-Boot code. Follow the procedures in the given board manual to write U-boot file.

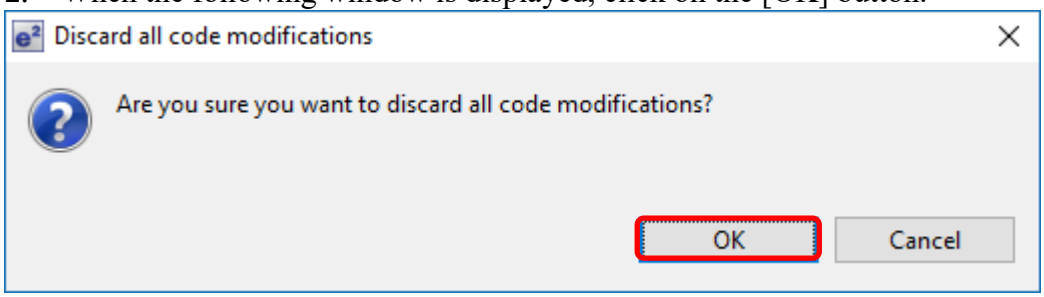
3.13.2 Removing the Contents of Editing the U-Boot Code

This subsection describes how to remove the contents of editing the U-Boot code.

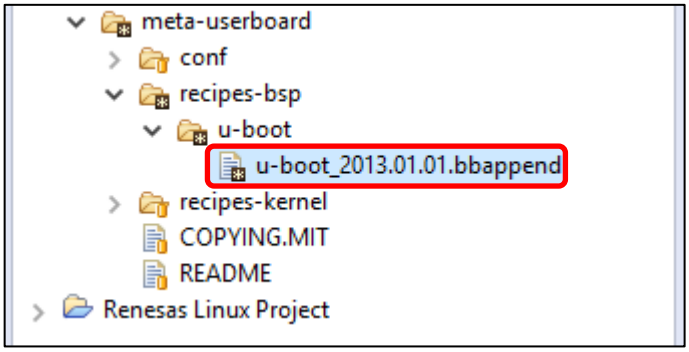
1. Right-click on the embedded Linux custom BSP project in the [Project Explorer] pane and select [Embedded Linux] -> [U-boot customization] -> [Discard modifications].



2. When the following window is displayed, click on the [OK] button.



3. [u-boot files] is removed from the embedded Linux custom BSP project in the [Project Explorer] pane.
   In addition, the Yocto recipe patch that has been automatically added during editing is also automatically removed and the code for applying the removal to the BSP is also automatically described.



4. To include the contents of removal as the build target, register the contents of editing with the Git. For how to register, see 3.14.2 Registering with the Git.
   To edit the U-Boot code again, download the U-Boot code again as described in 3.13.1 Editing the U-Boot Code.

3.13.3 Debugging the U-boot

To debug the U-boot, add instructions for outputting a desired character string to the console to the functions to be called by the U-boot and verify that the string is output to the console.

Debug the U-boot with the following procedure.

1. Add the following console output instruction before and after the target code to be debugged in each of the functions shown in 3.13.3.1.
   printf("argument 1");
   Argument 1: Specify a desired character string, register value, or variable.

2. Customize and download a binary file with the procedure described in 3.13.1.

3. Write the binary file to SPI flash memory by using ICE.
   The top address is the value of the "USER_PROGRAM_START_ADDRES" specified in "board_config.h". Check before debugging.

4. Start up the U-boot and debug by checking the console outputs.
   U-boot is automatically executed after transferred U-boot binary to DDR from SPI flash by the boot loader.

3.13.3.1 Functions to be Called when Executing the U-boot

After the U-boot has started up, it calls the following functions in order.

1. board_init_f()

   ./arch/arm/lib/board.c
2. arch_cpu_init()

   ./arch/arm/cpu/armv7/rmobile/cpu_info.c
3. board_init_r()

   ./arch/arm/lib/board.c
4. board_init()

   ./board/renesas/iwg20m_q7/iwg20m_q7.c *

Note: * The folder name "iwg20m_q7" and the source file name "iwg20m_q7.c" may differ depending on the BSP to be built.

The folder and source file names are determined by deleting "_config" from the character string specified for "UBOOT_MACHINE" in the local.conf file used for building the BSP.

The above shows an example where UBOOT_MACHINE is specified as follows:

UBOOT_MACHINE = "iwg20m_q7_config"

For the local.conf file, refer to section 3.17.

3.14 Git

Git is a control system of the files, for example, a source code file. The Git controls the update history, and thus has functions such as merging modifications to a single file.
This section describes how to use Git on the build server.
For details on how to use Git, refer to the following URL.
http://www.eclipse.org/egit/

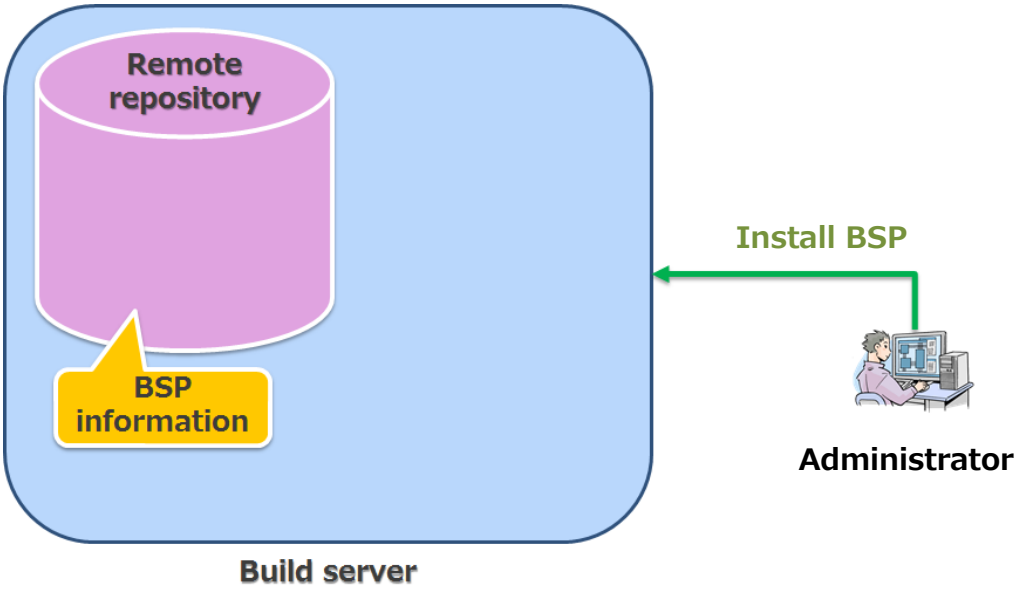
3.14.1 Using Git in the Version-control of Files

This subsection describes how to use Git in the version-control of files.

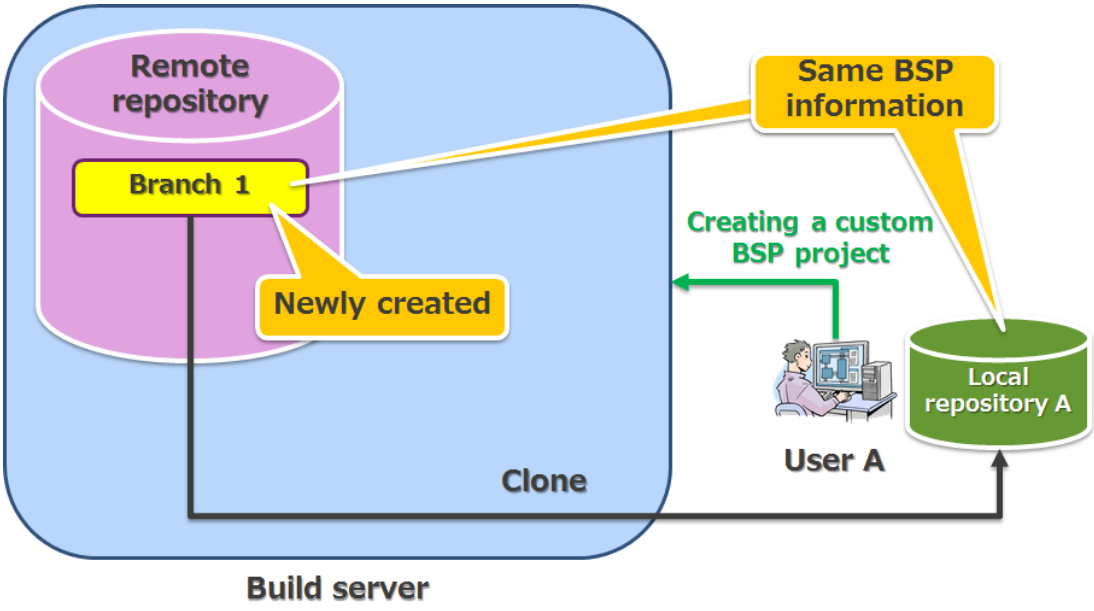
Note: Descriptions in this subsection are on the assumption that user A and B are using the same build server.

1. Details of the remote repository, local repository, and branch

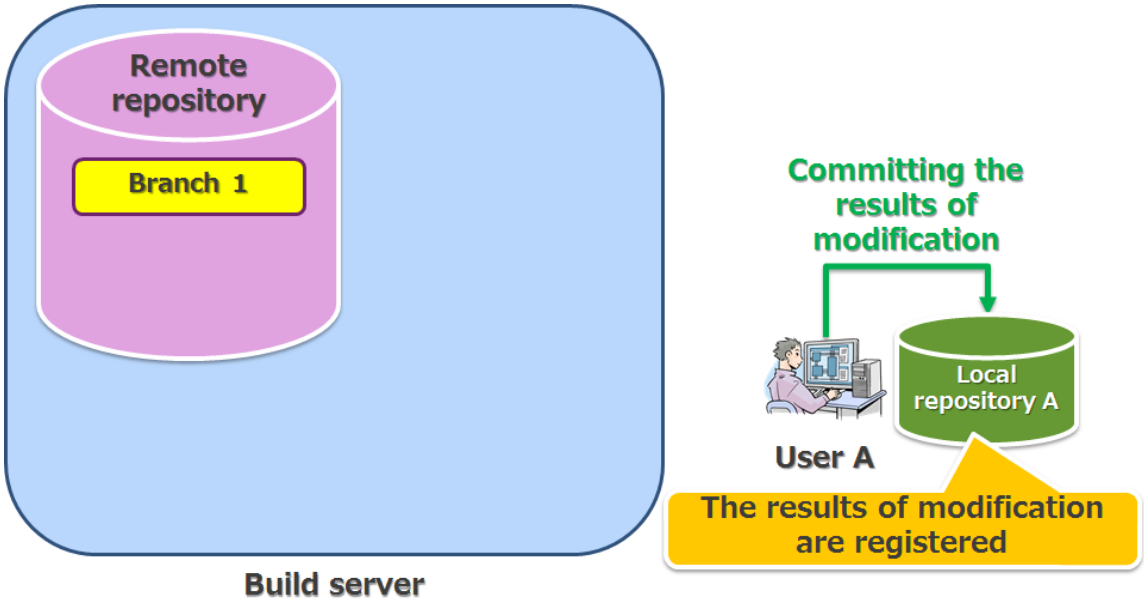
(1) When an administrator user ("admin") installs the BSP by operating on the Web screen of the build system, a remote repository is created on the build server. The installed BSP is registered with the remote repository.



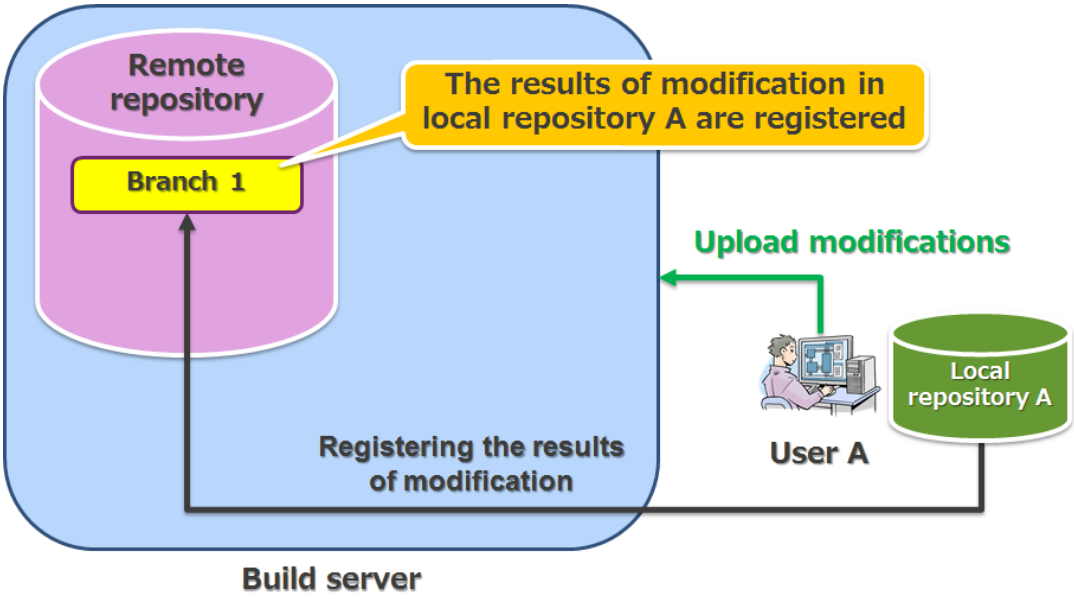
(2) Next, user A creates an embedded Linux custom BSP project.
At this time, if the remote repository has no branches, user A creates a new branch (hereinafter referred to as branch 1). Branch 1 contains the same BSP information as is registered with the remote repository.
In addition, a clone of branch 1 is created in the local repository (hereinafter referred to as local repository A) on the client PC of user A.
At this stage, branch 1 and local repository A contain the same BSP information.



(3) After that, user A customizes the embedded Linux custom BSP project and then commits the results of customization to register the results with local repository A.

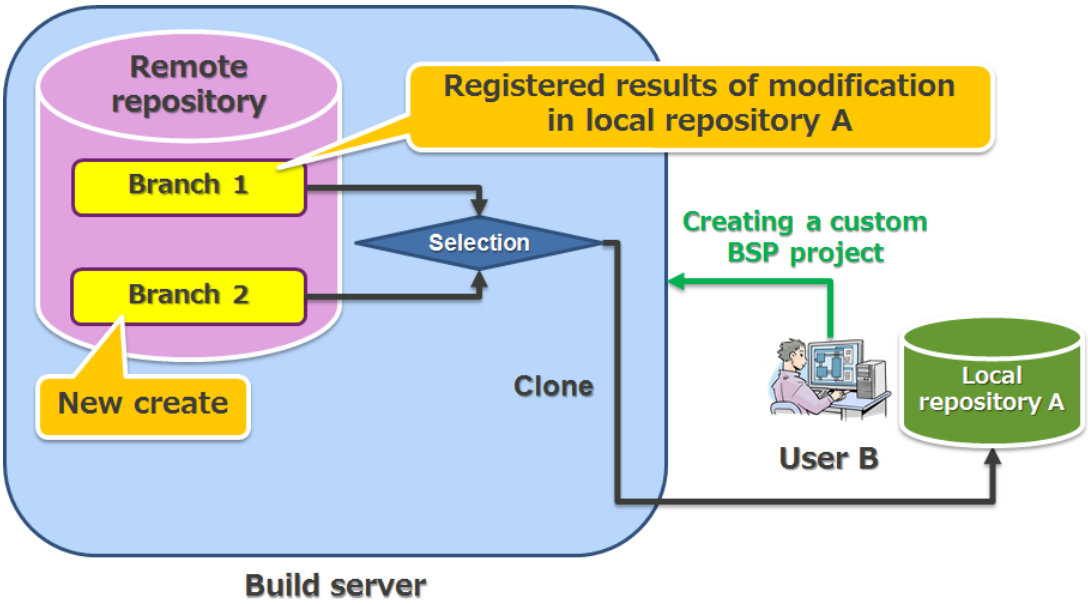


(4) If user A uploads further results of customization, the results in local repository A will be registered as the latest information in branch 1.



2. Sharing files between users

(1) When user B is to create an embedded Linux custom BSP project, user B can select either the latest information in the existing branch (branch 1 created by user A in this case) or a new branch (branch 2 in this case) as the target for cloning.
If user B selects branch 1, user B can share the results of customization by user A.

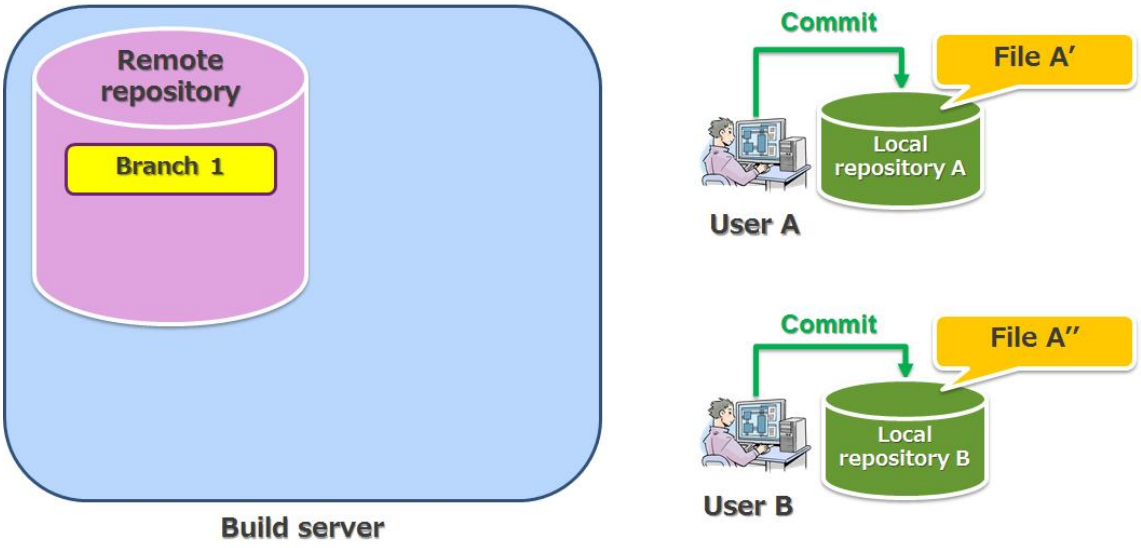


3. Conflicts between the results of customization by users and how to prevent them

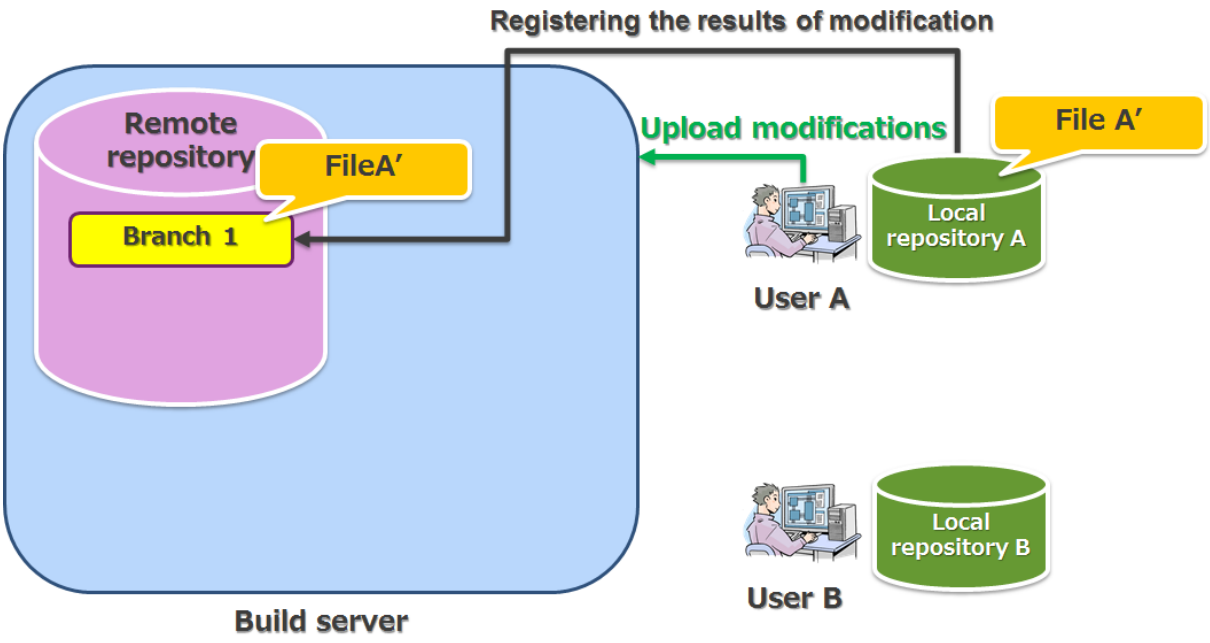
In the case of development by multiple users, conflicts (conflicts between the results of editing) may arise. Therefore, be sure to update the repository before committing a file to the local repository.

The following is an example of a conflict where we take the results from editing of File A by user A and user B as File A′ and File A″, respectively.

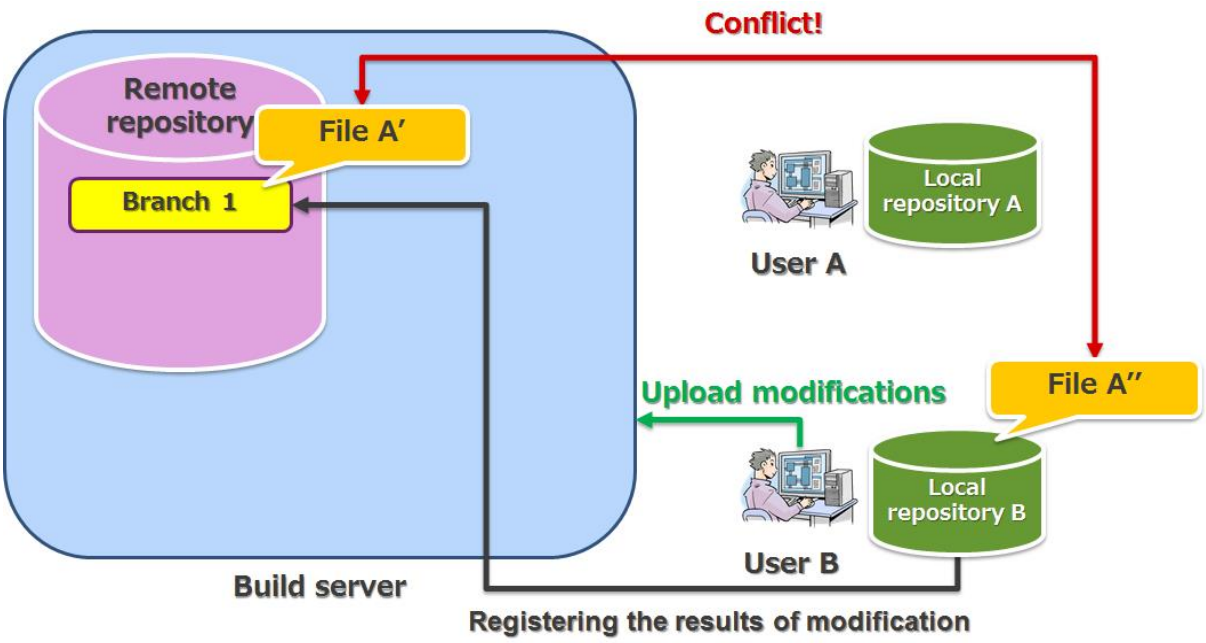
(1) Commits are registered in the local repositories of individual users. Therefore, if users only edit files in their own local repositories, no conflicts will arise.



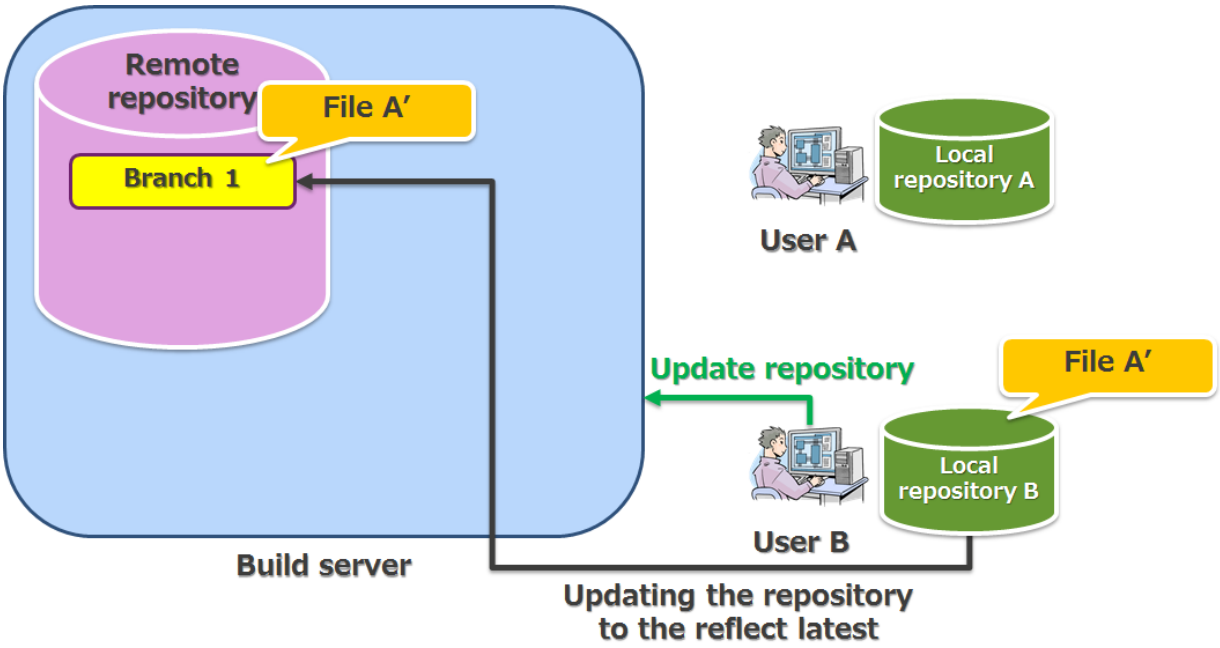
(2) Next, user A uploads the results of customization and registers File A′, the results of customization in local repository A, to branch 1 of the remote repository.



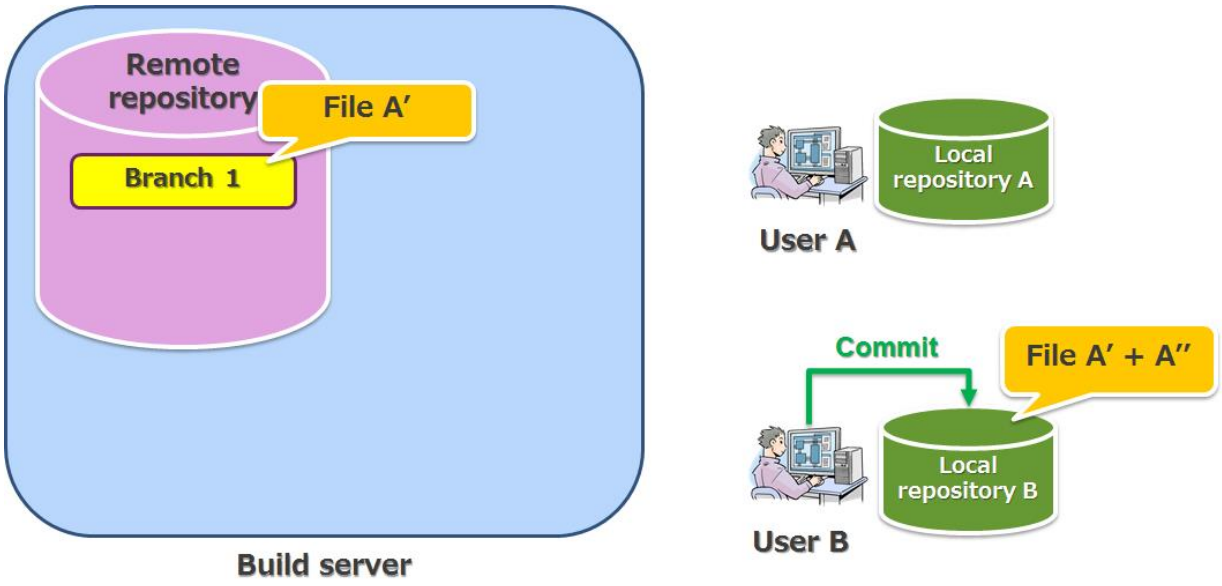
(3) After that, user B uploads the results of customization and attempts to register File A″, the results of customization in local repository B, in branch 1 of the remote repository. However, as File A′ has already been registered in branch 1 at this time, a conflict arises (conflict between the results of editing).



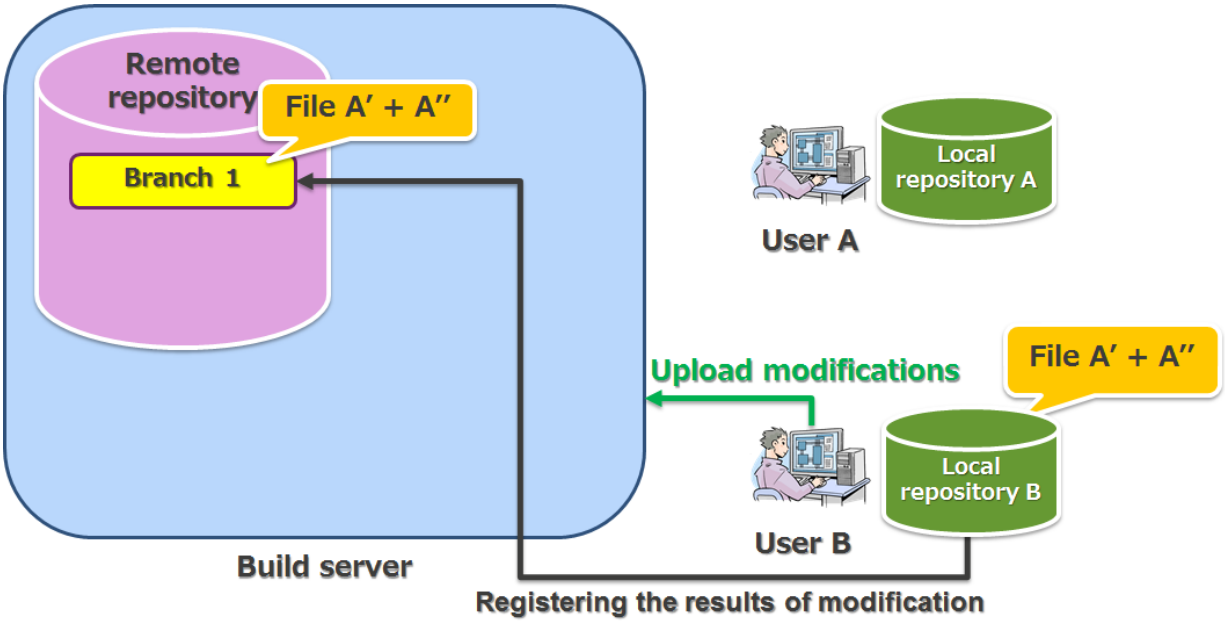
(4) To prevent or resolve conflicts, users must update their own repositories before committing the results of editing to keep the local repositories in the latest state.
In the figure below, user B updates local repository B to incorporate File A′, the result of editing by user A, in local repository B.



(5) Next, user B reflects his own contents of customization in File A′ by hand and commits File A′ + A″ in local repository B.



(6) Finally, user B uploads the results of customization and registers File A′ + A″, the contents of customization in local repository B, in branch 1 of the remote repository.
In this case, as File A′ + A″ contains the results of editing File A′ which were registered in branch 1, no conflicts will arise.

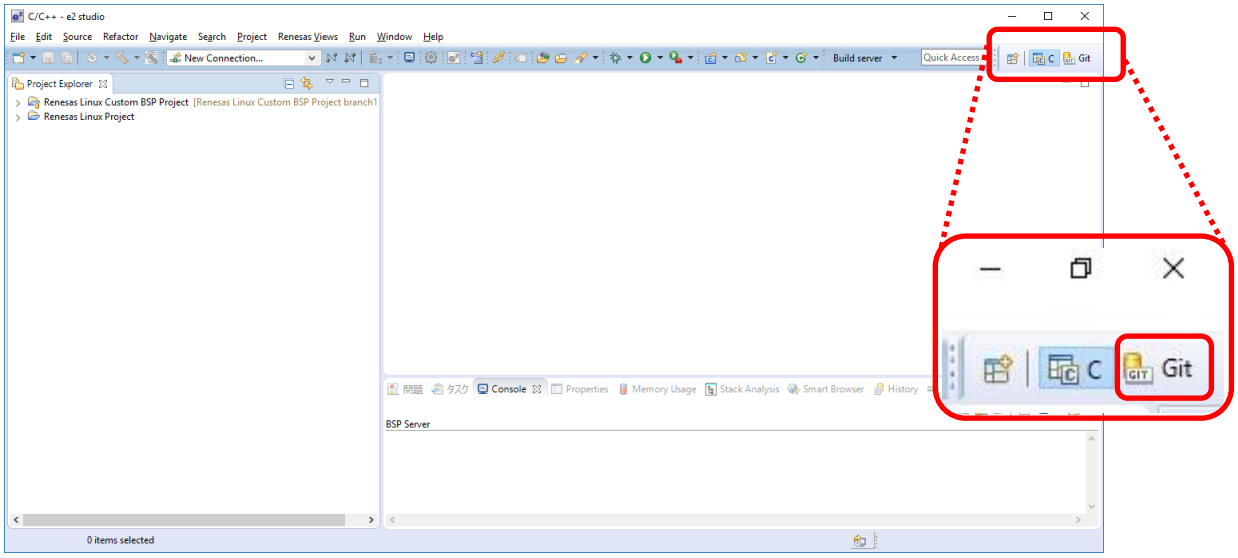


As described in steps 1 to 6, when proceeding with development by multiple users, be sure to prevent conflicts by updating your repository before committing files to the local repository.

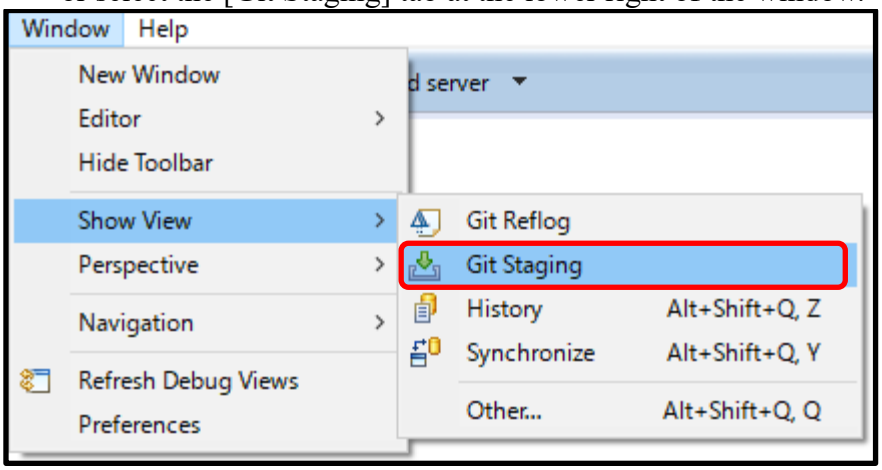
3.14.2 Registering with the Git

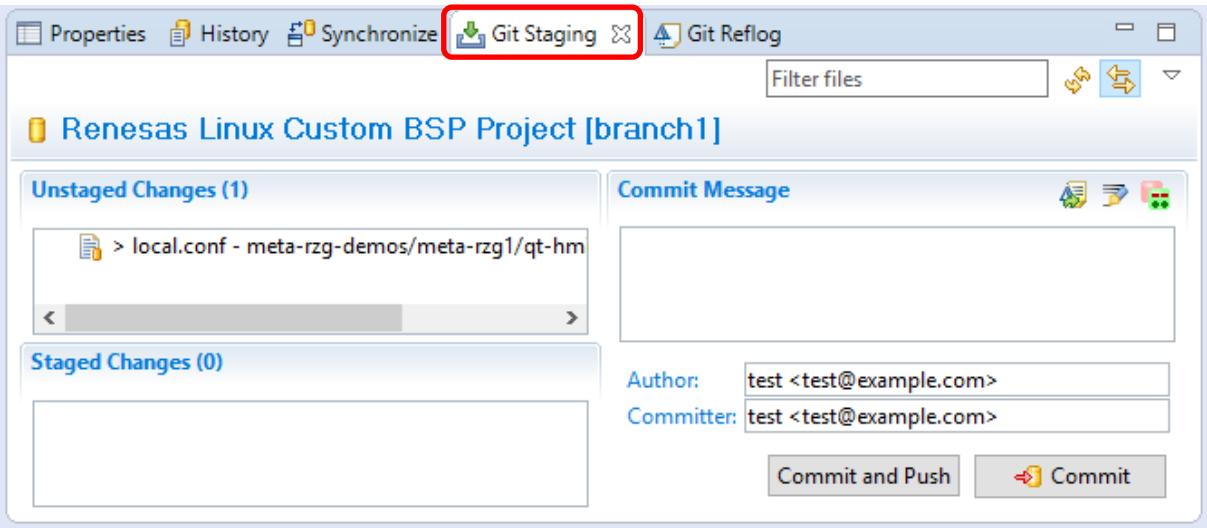
This subsection describes how to register the contents of customization of the project created in the embedded Linux custom BSP project with the Git.

1. Click on the [Git] button at the upper right of the window to open the Git repositories.

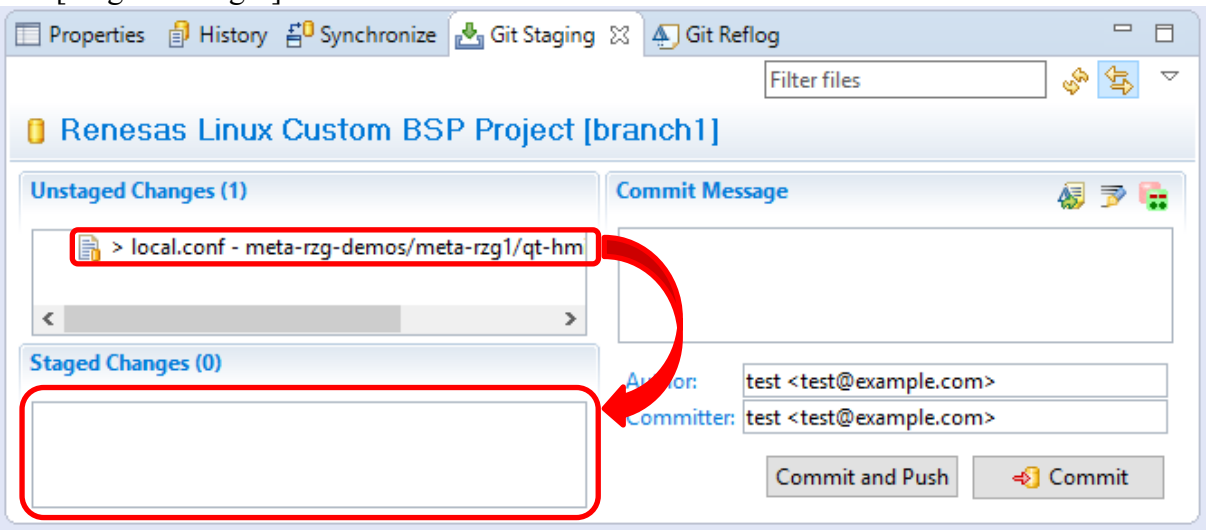


2. Select [Window] -> [Show View] -> [Git Staging] from the menu bar at the top of the window, or select the [Git Staging] tab at the lower right of the window.

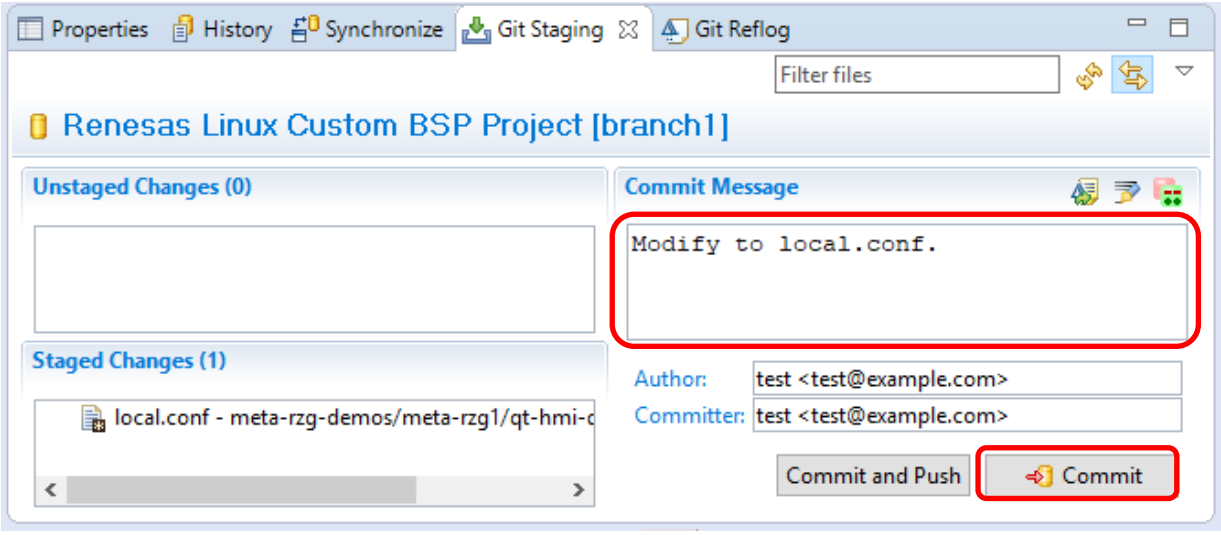




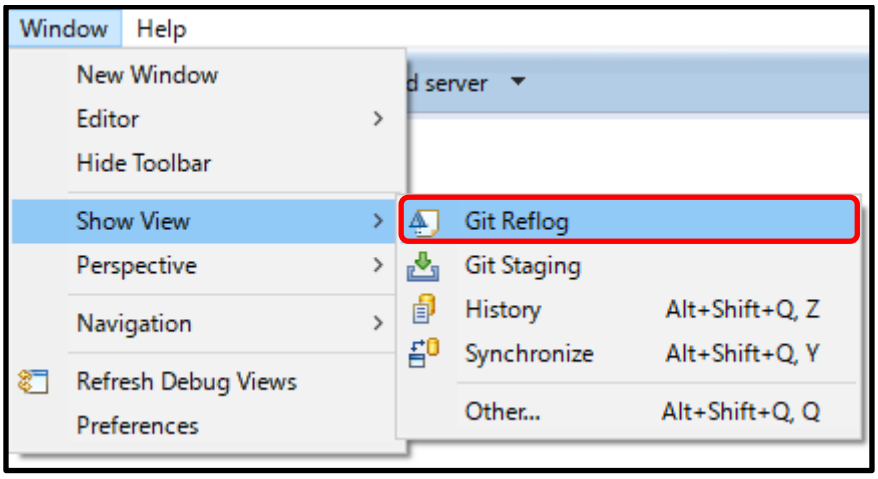
3. A set of files that have been customized (edited or added) is displayed in the [Unstaged Changes] box on the [Git Staging] tabbed page. Confirm the contents and drag and drop the files to the [Staged Changes] box.

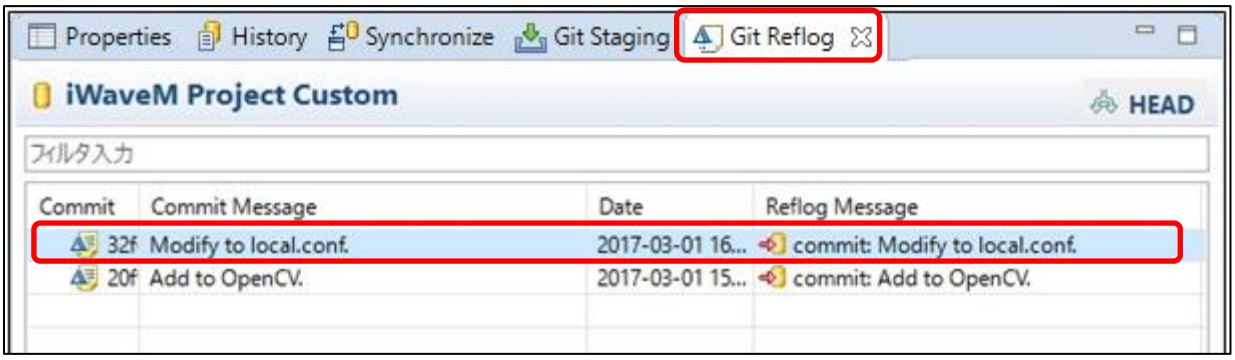


4. Enter the contents of customization (editing or addition) that have been made so far in the [Commit Message] box on the [Git Staging] tabbed page, and click on the [Commit] button. When Committed, the results of customization are registered with the local repository.



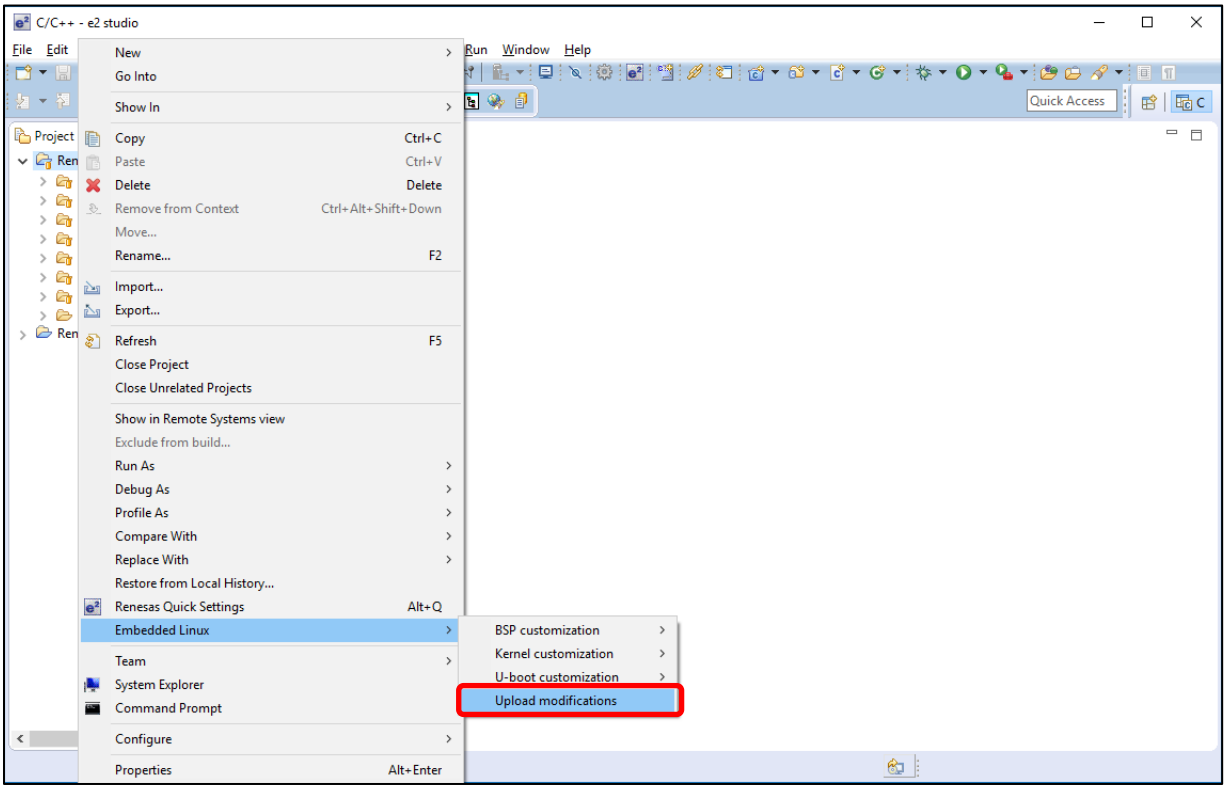
5. Select [Window] -> [Show View] -> [Git Reflog] from the menu bar at the top of the window, or select the [Git Reflog] tab at the lower right of the window. Confirm that the contents of customization entered are reflected.





6. To execute the build process according to the contents committed, select [Custom BSP project on the workspace] on the [BSP] tabbed page and execute the build process as described in 3.8.1 Setting up Build.

7. To register the results committed to a branch in the remote repository on the build server, right-click on the embedded Linux custom BSP project in the [Project Explorer] pane and select [Embedded Linux] -> [Update modification].



3.15 Creating a Custom BSP

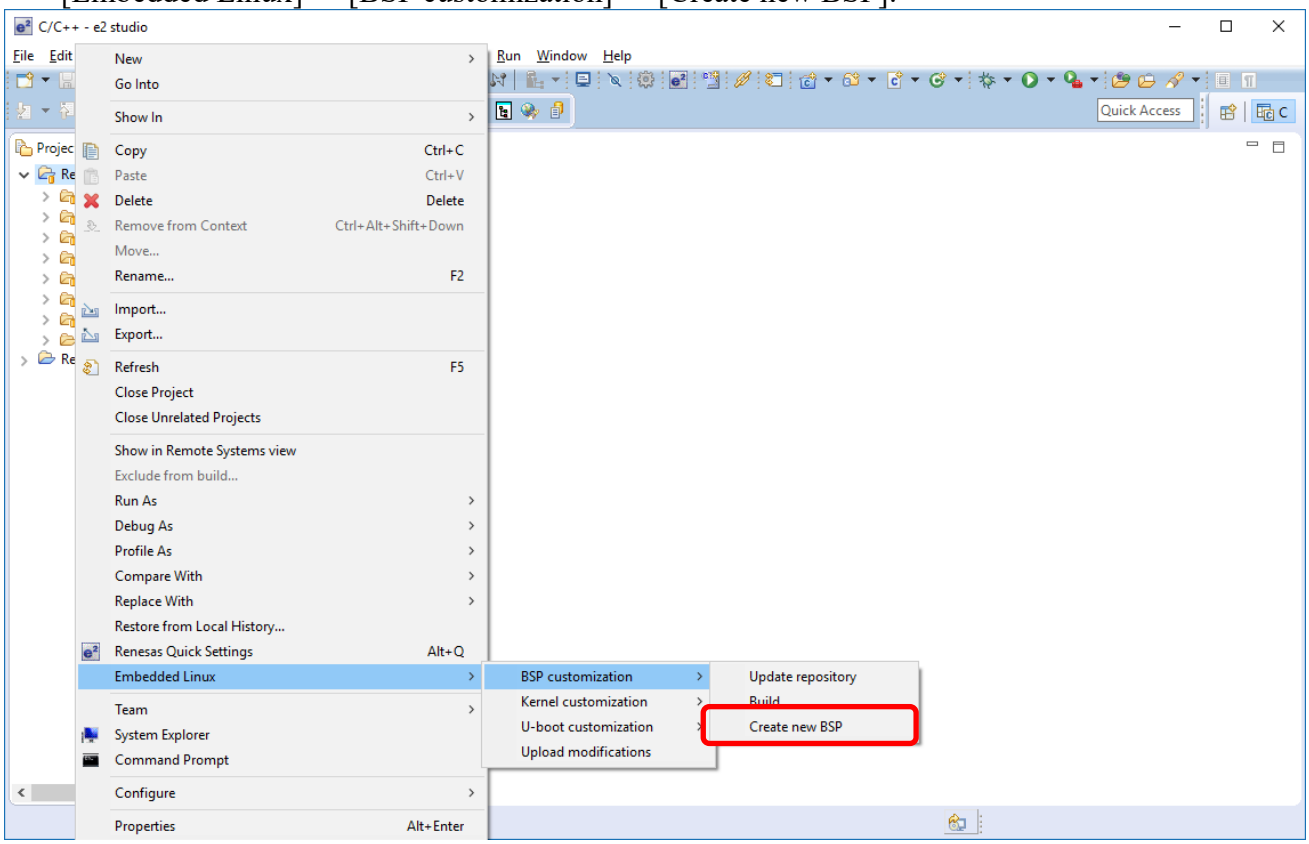
This section describes how to use the results of customization of an embedded Linux custom BSP project to create a custom BSP with the name which the user desires on the build server.
Note that the custom BSP is created based on the latest contents of the corresponding branch in the remote repository on the build server. For details on the remote repository and branches, see 3.14.1 Using Git in the Version-control of Files.

3.15.1 Creating a new BSP

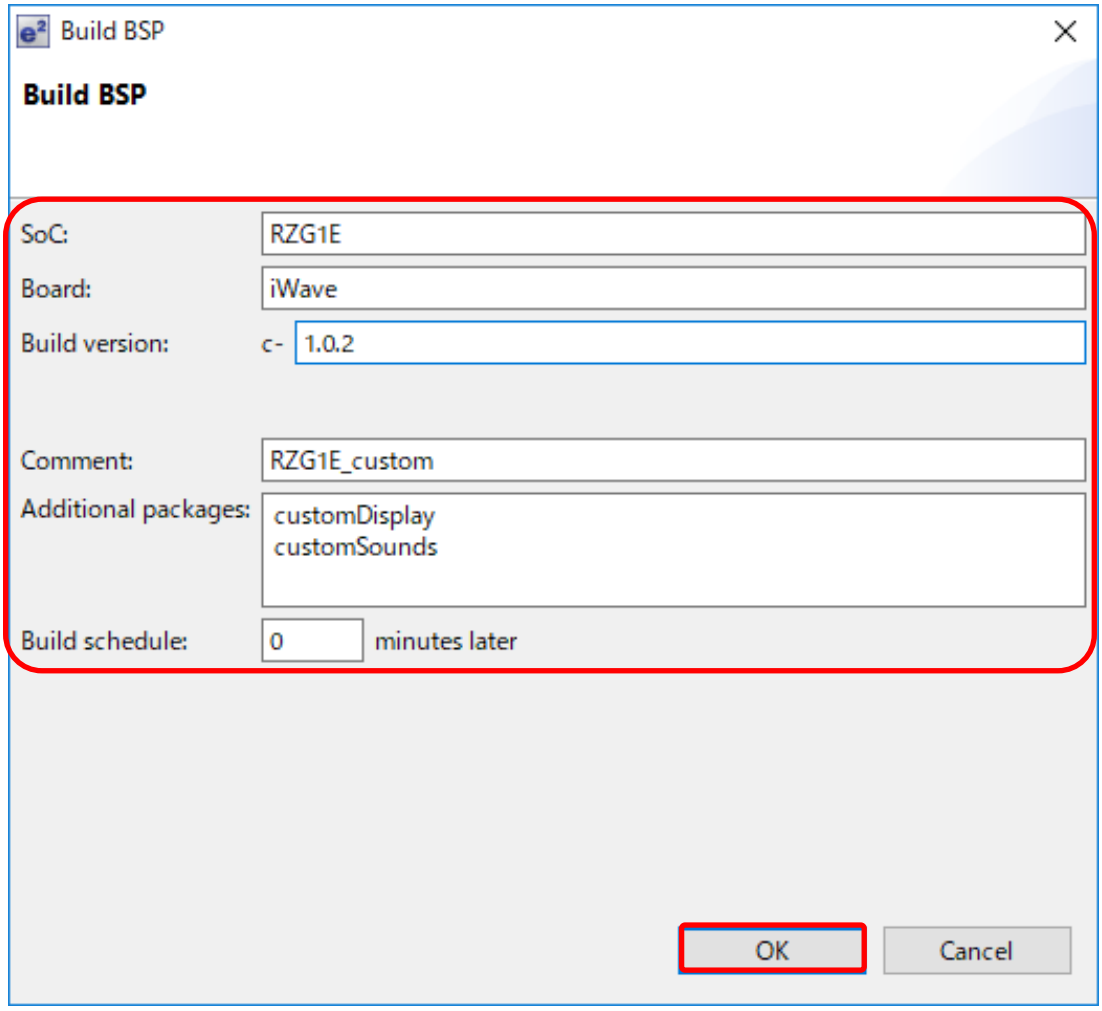
This subsection describes how to use the results of customization of an embedded Linux custom BSP project to create a new custom BSP on the build server.

Note: The BSP to be created is based on the latest contents of the corresponding branch (head) in the remote repository on the build server.
For details on the remote repository and branches, see 3.14.1 Using Git in the Version-control of Files.

1. Right-click on the embedded Linux custom BSP project in the [Project Explorer] pane and select [Embedded Linux] -> [BSP customization] -> [Create new BSP].

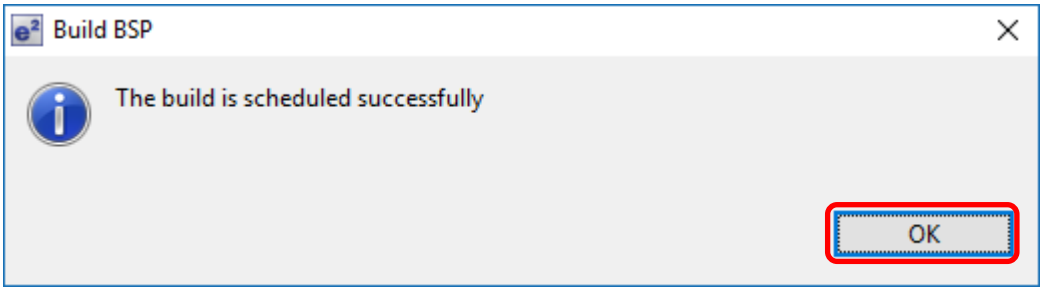


2. The [Build BSP] window appears. Enter all parameters and click on the [OK] button for creation of the BSP.
   Note: If you have a package to be added in the component list or the SDK, enter the Yocto recipe names (*1) in the [Additional Packages] section.
   (*1) The "name" part of the bb filename represented in the form name_version.bb



Parameter	Description
Soc	This parameter is used as part of the name of the BSP to be created. The chip name set here is used to specify the created BSP.
Board	This parameter is used as part of the name of the BSP to be created. The board name set here is used to specify the created BSP.
Build version	This parameter is used as part of the name of the BSP to be created. The version set here is used to specify the created BSP. The Build version is automatically prefixed with "c-".
Comment	Enter the description of the BSP to be created. The entry in [Comment] is displayed in various windows such as the section for selecting the BSP in the dialog box for setting up the build process.
Additional Packages	Specify parameters here to add packages (Yocto recipes). The packages entered here are of the following types and added after creation of the BSP. • User add-ons when standard components are selected • SDK
Build Schedule	This parameter specifies when to build the custom BSP. Specifying 0 leads to the build process starting immediately. If you wish to proceed with the build process when the load on the build server is light, for example at night, specify an appropriate time in minutes.

3. When the build is automatically started and finished, the completion window appears. Click on the [OK] button.
   Note: The new BSP will be created in a few hours after the complete window appears.
   You cannot go next step until the new BSP is created.



4. After successful creation of the BSP, you can use the new BSP in e^2 studio by specifying [Soc], [Board], [Build version] entered in the Build BSP window.

3.16 Creating and Changing an SDK

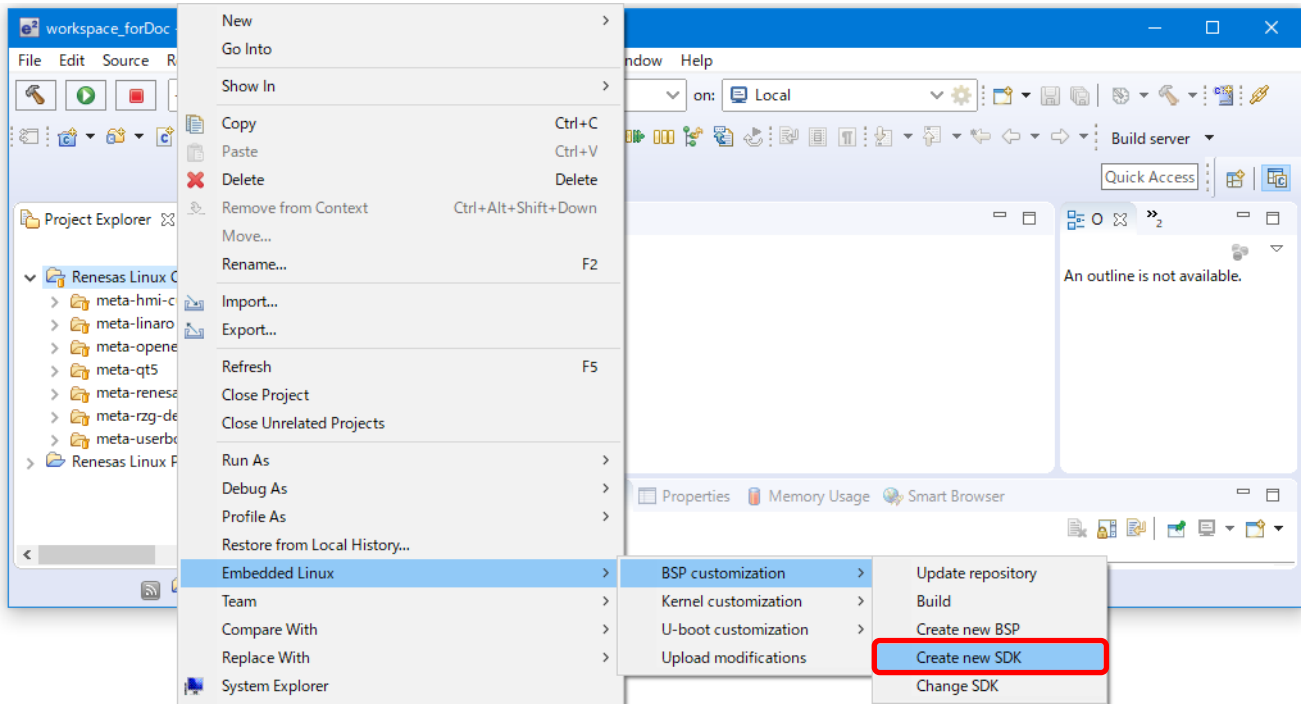
This section describes how to create new SDK by embedded the library file for using package added in Yocto recipe and use it.

For how to Yocto recipe adding, see 3.11.3 Adding Yocto Recipe.

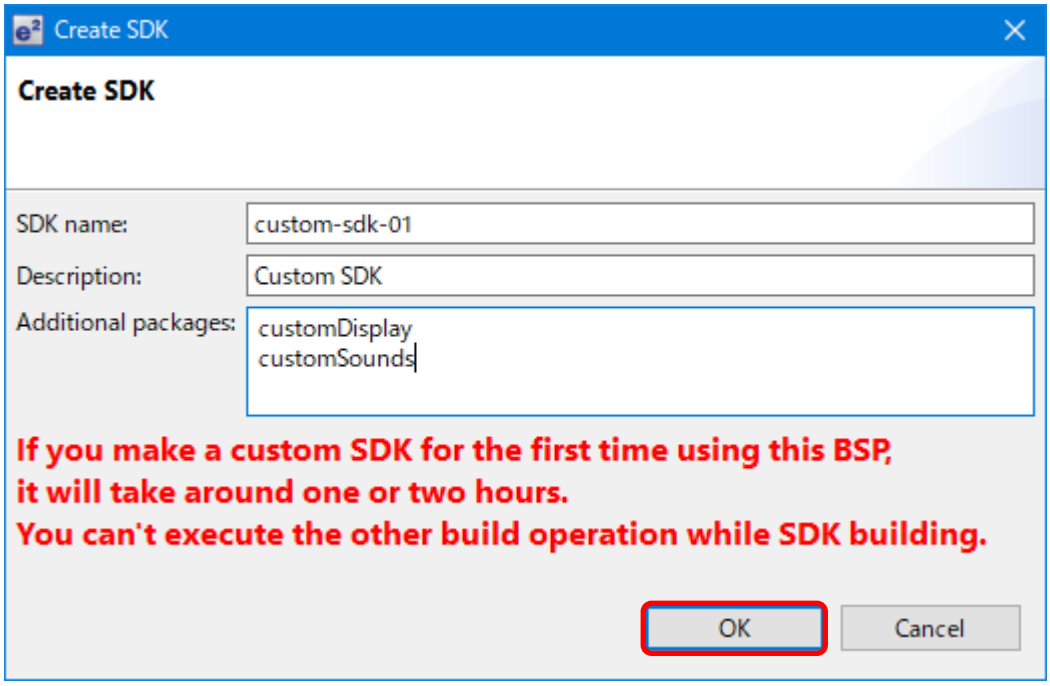
3.16.1 Creating a new SDK

This subsection describes how to create new SDK embedded the package added in Yocto recipe to original SDK in BSP.

1. Right-click on the embedded Linux custom BSP project in the [Project Explorer] pane and select [Embedded Linux] -> [BSP customization] -> [Create new SDK].



2. The [Create SDK] window appears. Enter all parameters and click on the [OK] button for creation of the BSP.
   Note: If you have a package to be added in SDK, enter the Yocto recipe names (*1) in the [Additional Packages] section.
   (*1) The "name" part of the bb filename represented in the form name_version.bb



[Point for Caution on Setting the new SDK name]

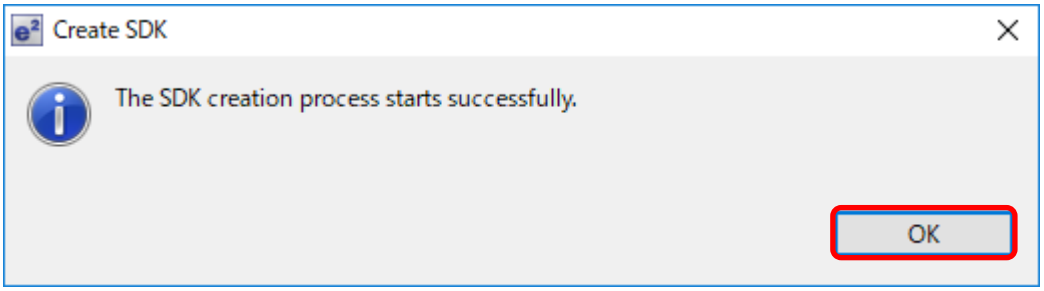
If you set an already created SDK name to the new SDK name, an already created SDK is overwritten by new SDK.

Note: the original SDK name in BSP is "original". If you set the new SDK name to "original, the original SDK is overwritten by new SDK.

[Point for Caution on Setting the Additional Package name]

You cannot add the package to the new SDK after created it, because [Additional Packages] is always add to the original SDK in BSP. You must set the all recipe name you need to [Additional Packages] when creating a new SDK.

3. When the build is automatically started and finished, the completion window appears. Click on the [OK] button.
   Note: The new SDK will be created in a few hours after the following window appears.
   You cannot go next step until the new SDK is created.



4. After successful creation of the SDK, you can use the new SDK by described in 3.16.2 Changing a SDK.

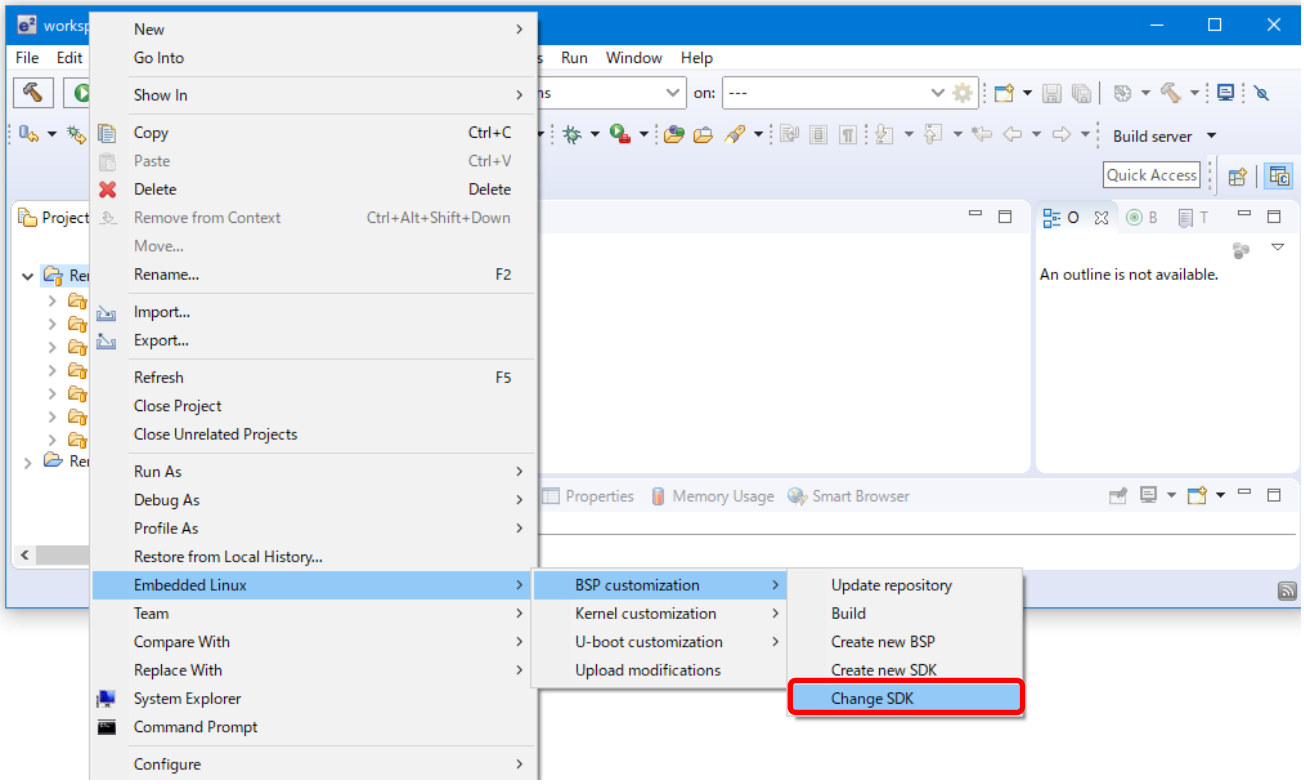
[Point for Caution on Creating the new SDK]

The created new SDK can be shared among users. If you want to use SDK created by another user, change the SDK refer to 4.16.2 Changing a SDK.

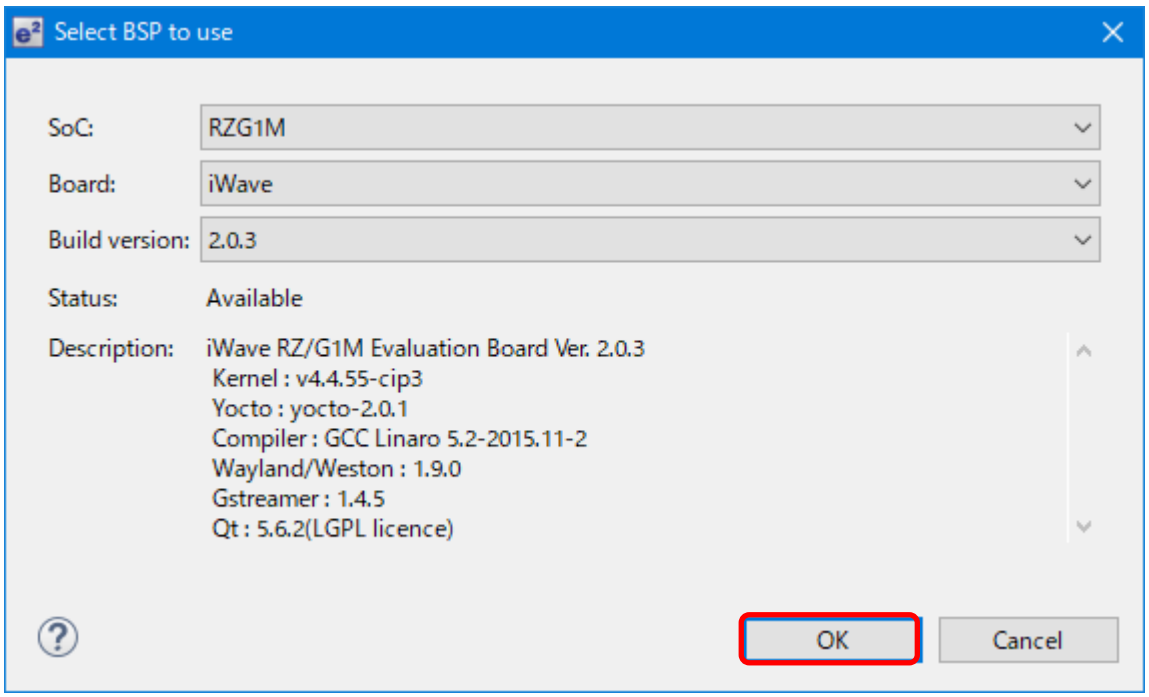
3.16.2 Changing an SDK

This subsection describes how to change a SDK used for build.

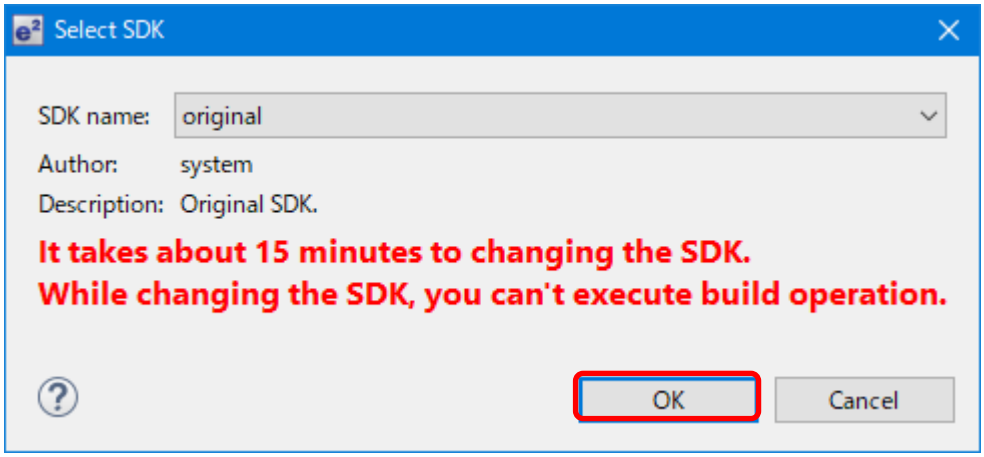
1. Right-click on the embedded Linux custom BSP project in the [Project Explorer] pane and select [Embedded Linux] -> [BSP customization] -> [Change SDK].



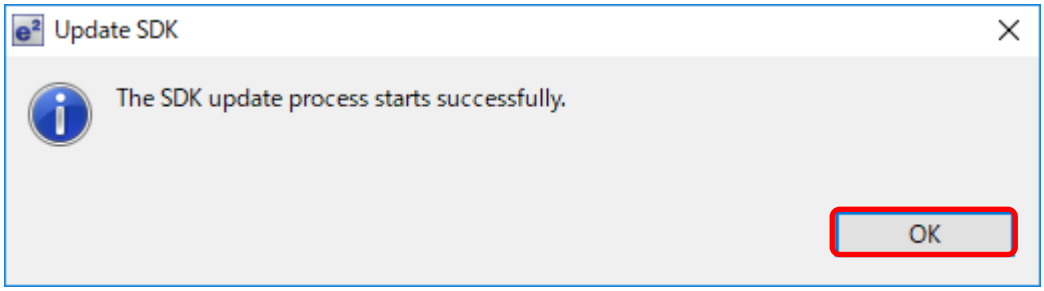
2. Select the BSP to change the SDK on the [Select BSP to use] window and clock the [OK] button.



3. The [Select SDK] window opens. Select the SDK name you want to change from the [SDK name] pull-down menu.
   The "original" in pull-down menu is an original SDK in BSP.



4. After successful changing the SDK, the following window is displayed, and Clock the [OK] button.
   Note: The SDK will be updated in about fifteen minutes after the following window appears.

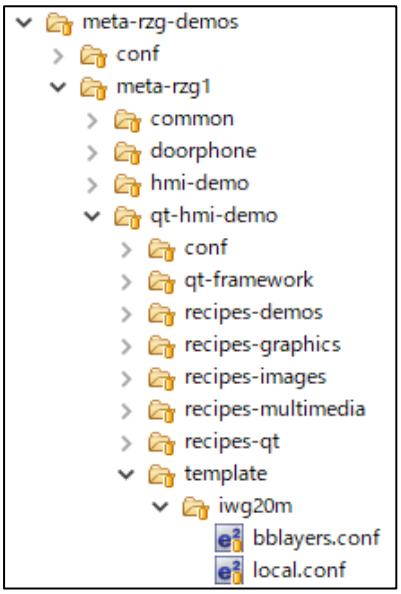


3.17 Editing the local.conf

This section describes how to edit the local.conf. The local.conf is a file for specifying CPU architecture and MACHINE Configuration (Example: Customer specific string such as product name).

1. Create an embedded Linux custom BSP project to download the local.conf.
   For how to create an embedded Linux custom BSP project, see 3.5.2 Creating an Embedded Linux Csutom BSP Project.

2. Open and edit the local.conf to be edited. The local.conf is stored in the following directory.
   \meta-rzg-demos\meta-rzg1\qt-hmi-demo\template\<MACHINE Configuration>\



Relationship between the chip name and Machine Configuration is refer to Table 3-1.

Table 3-1. Relationship between the chip name and Machine Configuration

Chip name	BSP version	MACHINE Configuration
RZ/G1E	1.0.0	iwg22m
	1.2.1	
	2.x.x	
RZ/G1N	1.0.0	iwg20m
	1.2.1	iwg20m-rzg1n
	2.x.x	iwg20m
RZ/G1M	1.0.0	iwg20m
	1.2.1	iwg20m-rzg1m
	2.x.x	iwg20m
RZ/G1H	1.0.0	iwg21m
	1.2.1	
	2.x.x	

3. When save the edited local.conf and execute build, it is reflected in the result of build.

[Point1 for Caution 1 on Editing the local.conf]

In the local.conf, the following "MACHINE Configuration" is described.

Example) MACHINE ??= "iwg20m"

When "MACHINE Configuration" is changed, it is necessary to change all files and directories using "MACHINE Configuration". For example, change the reference destination of the DTB file, create a new DTB file, etc.

If create a new DTB file, it is necessary to name it according to the following rule.

<Any name>-<MACHINE Configuration>.dts

Note: When editing local.conf and executing the build, there is a possibility of taking several hours depending on editing content.

[Point for Caution 2 on Editing the local.conf]

Edit the existing local.conf stored in following directory because the local.conf cannot be newly added.

\meta-rzg-demos\meta-rzg1\qt-hmi-demo\template\<MACHINE Configuration>\

Note: The name of directory in which local.conf is stored cannot change. if you change it, the result of build will be an error.

4. Handling the Verification Tool

The Verification tool can be used only when purchased.

4.1 Starting up the Verification Tool

Select [Show View] -> [Other] from the [Window] menu on the e^2 studio. The screen is displayed as shown in Figure 4-1.
On the screen, select [RZ/G Linux Platform tools] -> [RZ/G Verification items and execution]. The verification tool window is displayed as shown in Figure 4-2.
Note that all the menu items described in this section refer to those in the window shown in Figure 4-2.

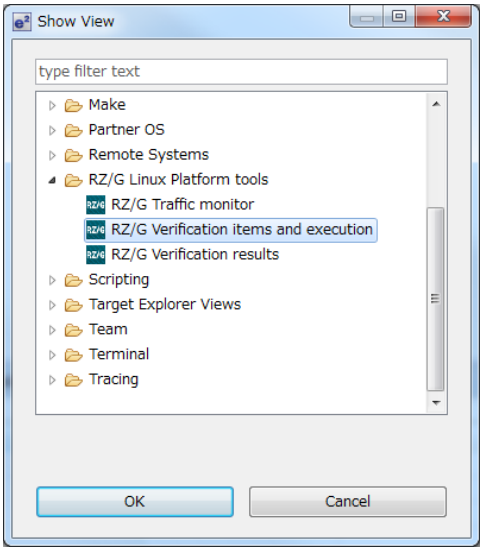


Figure 4-1 Screen for Selecting Window to Display

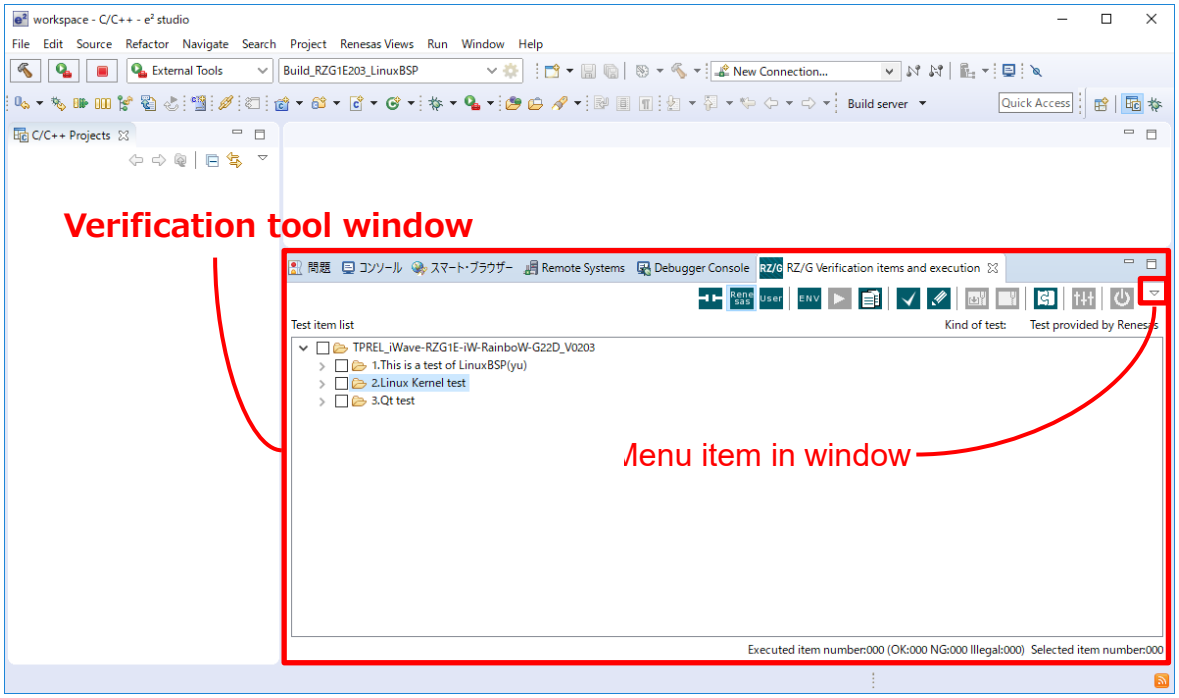


Figure 4-2 Verification Tool Window

4.2 Tool Settings

To use this tool, the following settings need to be made to suit the environment you are using.

1. Settings for connection with the target board
2. Specifying the destinations for storage of the test programs and results of tests
3. Selections for the test environments

Items to be set and their contents are described on the following pages.

4.2.1 Settings for Connection with the Target Board

Select [Preferences] from the [Window] menu on the e^2 studio, and the [Preferences] dialog box will open. Select [RZ/G Linux Platform tools] in the left pane and make the settings as shown in Table 4-1. Examples of the settings are shown in Figure 4-3,
Figure 4-4, and Figure 4-5.

Table 4-1 Tool Settings to Be Made in Relation to Connection with Target Board

Tab Name	Item to Be Set	Setting
Serial port	COM port	Specify the COM port on the host PC for serial connection to the Linux console port on the target board.
	Baud rate	Specify the communication parameters for the serial port according to the target board.
	Data bits	
	Parity	
	Stop bits	
	Flow control	
Network connection	Host name	Enter the network address of the target board.
	SSH Port	Enter the port number used by the target board for SSH connection.
	User name	Enter the network login ID.
	Password	Enter the network login password. Note: When SSH public key-based authentication is used, this setting is invalid.
	Secret key File	When SSH public key-based authentication is to be used for logging in, specify the private key file to be used.
Console connection	User name	Enter the login ID to be entered when executing test programs.
	Password	Enter the login password to be entered when executing test programs.

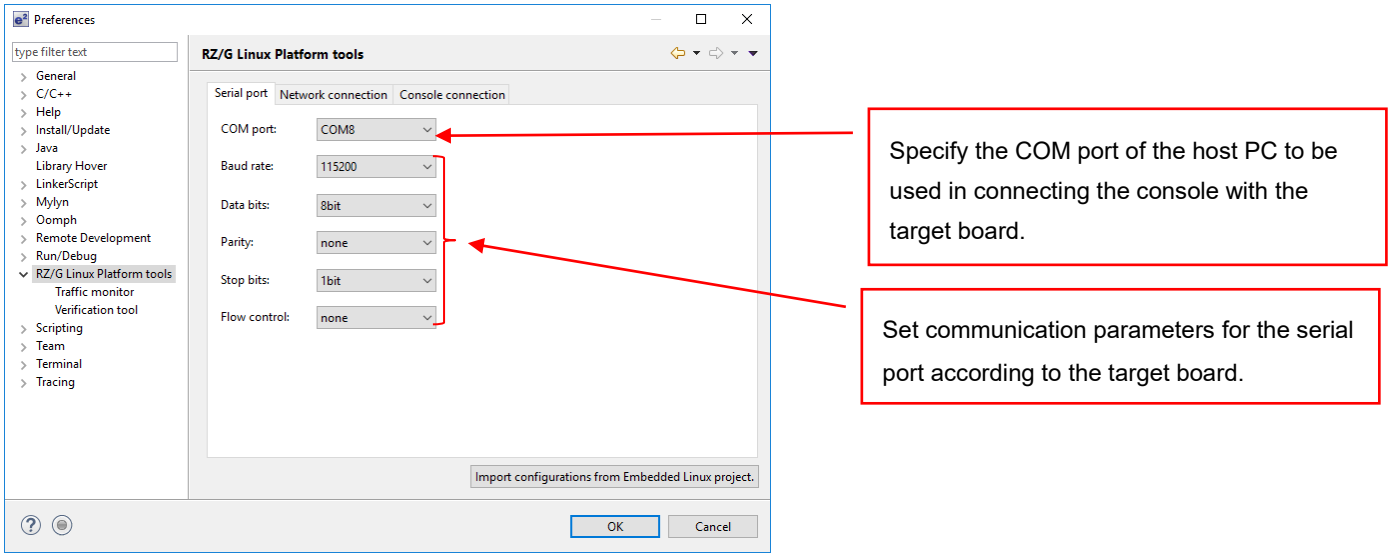


Figure 4-3 Example of Settings on [Serial port] Tabbed Page

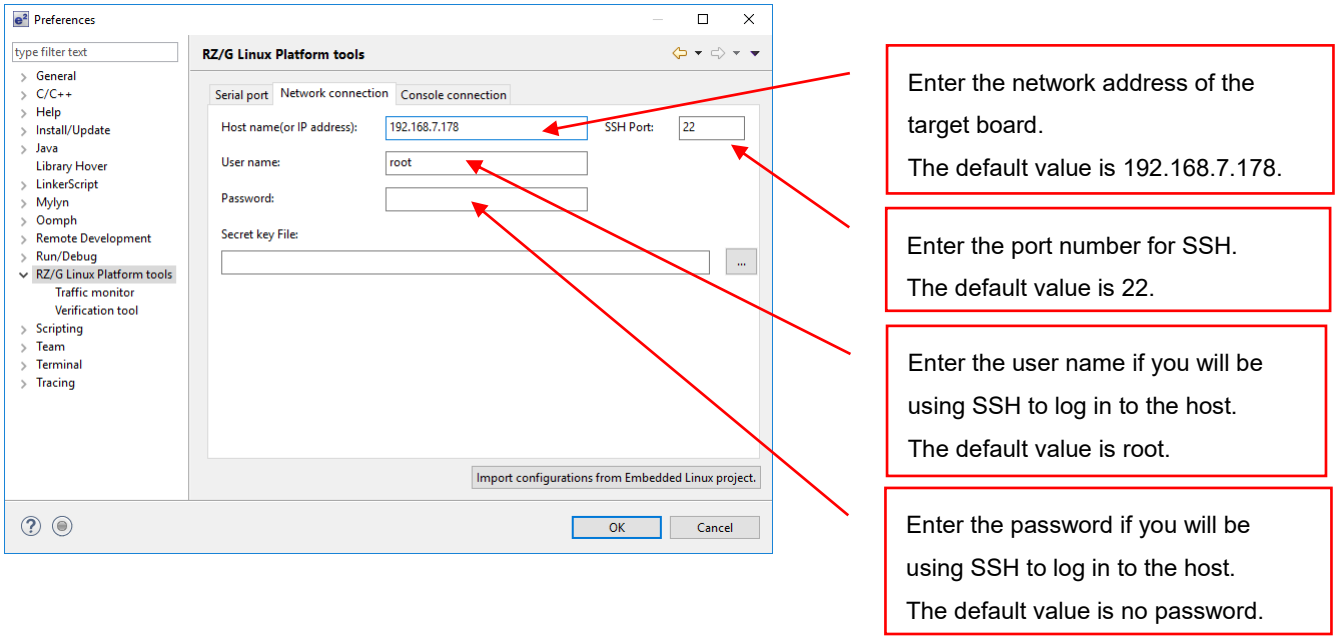


Figure 4-4 Example of Settings on [Network connection] Tabbed Page

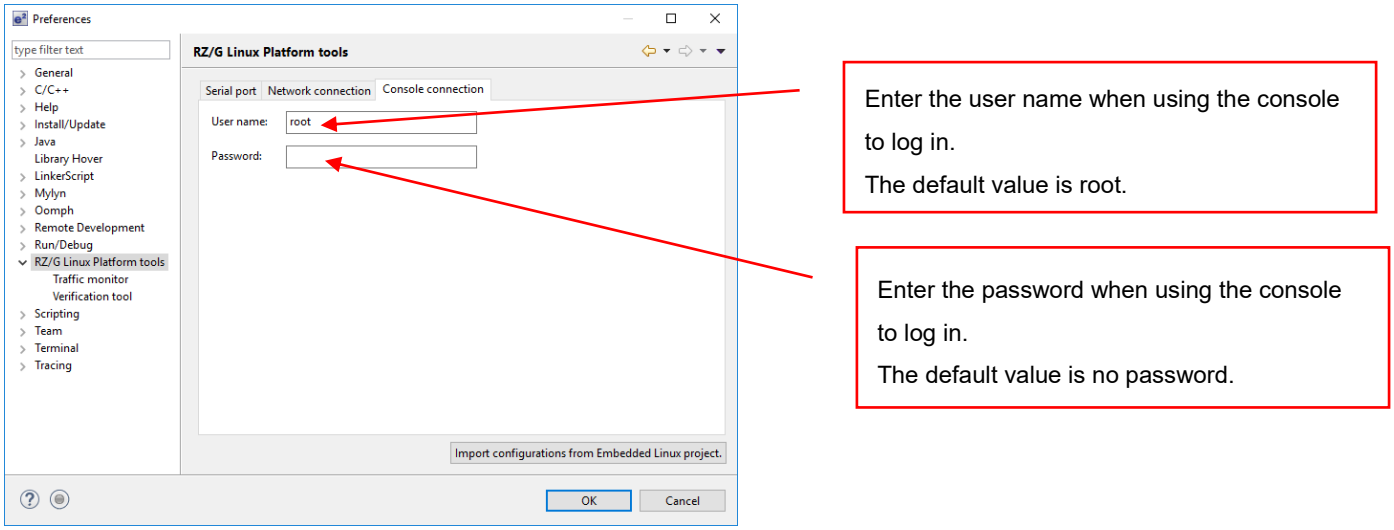


Figure 4-5 Example of Settings on [Console connection] Tabbed Page

4.2.2 Using the Embedded Linux Project to Import the Settings

Among the settings described in section 4.2.1, Settings for Connection with the Target Board, the Embedded Linux Project can be used to import those for the serial port and network connection.

Select [Preferences] from the [Window] menu on the e^2 studio, and the [Preferences] dialog box will open. Select [RZ/G Linux Platform tools] in the left pane and click on the [Import configurations from Embedded Linux project.] button.
Select the target project from the list as shown in Figure 4-6 and click on the [OK] button.

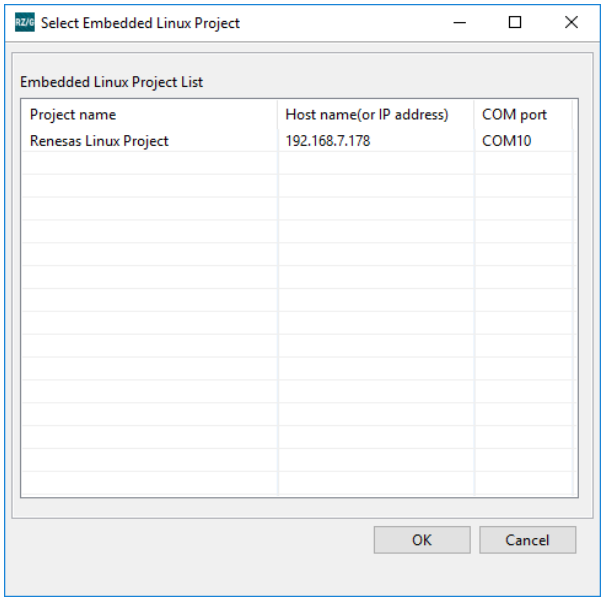


Figure 4-6 Screen for Selection of the Embedded Linux Project

4.2.3 Specifying the Destinations for Storage of the Test Programs and Results of Tests

Select [Preferences] from the [Window] menu of the e^2 studio, and the [Preferences] dialog box will open. Select [RZ/G Linux Platform tools] -> [Verification tool] in the left pane and make settings related to the execution of tests as described in Table 4-2.

Table 4-2 Tool Settings to Be Made in Relation to Test Programs

Tab Name	Item to Be Set	Setting
Running Tests	Folder to which test programs are to be transferred on the target board or an NFS server	Specify the folder to which test programs are to be transferred by using the SSH connection. *1 If tests are run without transferring test programs, specify the path for storing the test program on the target board.
	Folder for running test programs on the target board	Specify the folder from which test programs are to be run on the target board. *1
	Folder to which the imported test program for software add-on is to be loaded	This is the folder to which the imported test program for software add-on is to be loaded. *2
	Folder to edit parameter for test program	This is the folder holding edited parameters of test program. *2
	Folder holding test programs provided by Renesas	This is the folder holding test programs provided by Renesas. The default is C:\RZG-LinuxPF-Tools\RenesasTP. *2
	Folder to which test programs created by users are to be loaded	Specify any folder. This is the folder to which the tool loads test programs created by users when they are registered with the tool. *2
	Folder to hold the test results	This is the folder which the tool uses internally to hold the results of tests. *2
	When an error is found during execution of a test, the subsequent test does not proceed.	This specifies whether to terminate tests when an error is detected during a test.
	When an error is encountered during a test, the subsequent test does not proceed.	This specifies whether to terminate tests when an error (illegal operation by a program) is detected during a test.
	Tests proceed without transferring test programs.	This specifies whether to run tests without transferring the test program from the PC. *3
	Tests proceed through console connection or tests proceed through SSH connection.	This selects whether to run tests through console connection or network (Ethernet) connection. *4

Notes: 1. Specify the same folder unless you require the test programs to be transferred and run from folders with different names.
2. For this item, specify the path of the folder created in section 2.1.6. If a different folder has been created from the one described in section 2.1.6, specify the pertinent folder.
3. To run tests with this setting, it is necessary to load the test programs to the target board in advance and to specify its path as the "folder to which test programs are to be transferred on the target board or an NFS server." The test programs can be registered as an embedded Linux project and loaded to the target board in the same way as is described in section 4.8. Refer to section 4.8 for the required setting.
4. To run tests through console connection, it is necessary to load the test programs to the target board in advance and to specify its path as the "folder to which test programs are to be transferred on the target board or an NFS server." Refer to section 4.8 for the required setting.

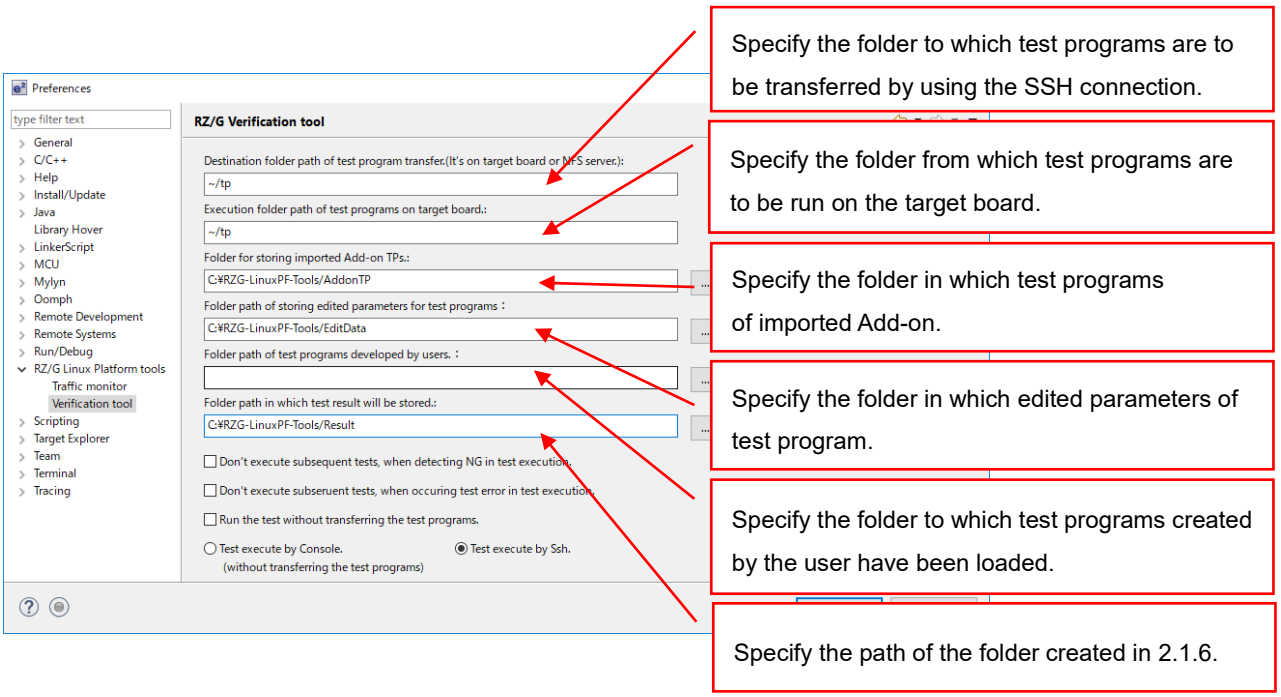


Figure 4-7 Example of Settings on [Test execution] Tabbed Page

4.2.4 Selections for the Test Environments

1. When test programs provided by Renesas are to be executed

Take either of the following steps to select a test environment which suits the target board.

(1) Select [Select kind of test] -> [Test provided by Renesas] from the [Test] menu, or click on the [Test provided by Renesas] button (the [Renesas] label) on the toolbar.

(2) Select the [Select target Environment] menu item from the [Test] menu, or click on the [Select target Environment] button (the [ENV] label) on the toolbar.

The list of the environments for the test programs in [Folder holding test programs provided by Renesas] is displayed in a drop-down list from the [Test provided by Renesas] text box in the dialog box shown in Figure 4-8. Select the suitable environment.

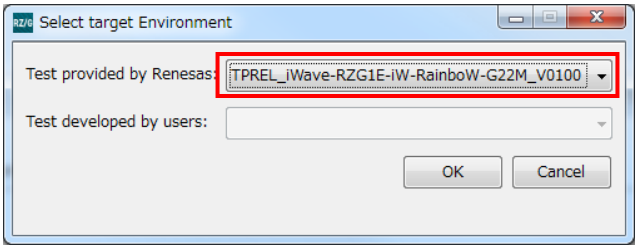


Figure 4-8 Selections for Test Environments (Test Programs Provided by Renesas)

2. When test programs created by users are to be executed

Go through the following procedure to select the environment in which the test is to be executed.

(1) Register a test program. For details on how to register a test program, see 4.6 Registering and Editing Test Programs Created by Users.

(2) Select [Select kind of test] -> [Test developed by users] from the [Test] menu, or click on the [Test developed by users] button (the [User] label) on the toolbar.

(3) Select the [Select target Environment] menu item from the [Test] menu, or click on the [Select target Environment] button (the [ENV] label) on the toolbar.

The list of the environments for the test programs in folder to which test programs created by users are to be loaded is displayed in a drop-down list from the [Test developed by users] text box in the dialog box shown in Figure 4-9. Select the suitable environment.

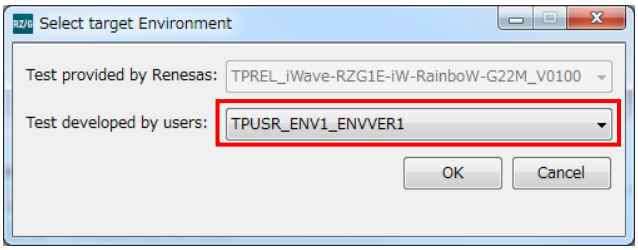


Figure 4-9 Selections for Test Environments (Test Programs Created by Users)

4.3 Executing Tests

4.3.1 Selecting Items to Be Tested

Select items to be tested from the [Test item list] section. The items are selected by clicking on the checkboxes for the corresponding nodes. Selecting an item (a parent node) with children leads to the selection of all items under the parent node.

An example of the selection of items is shown in Figure 4-10 and Figure 4-11. In this example, selecting item 1-1-4 selects items 1-1-4-1 and 1-1-4-2.

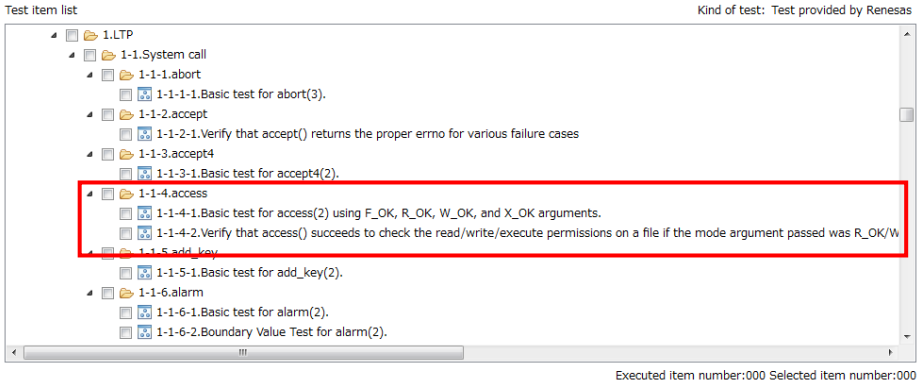


Figure 4-10 Before Selecting Items to Be Tested

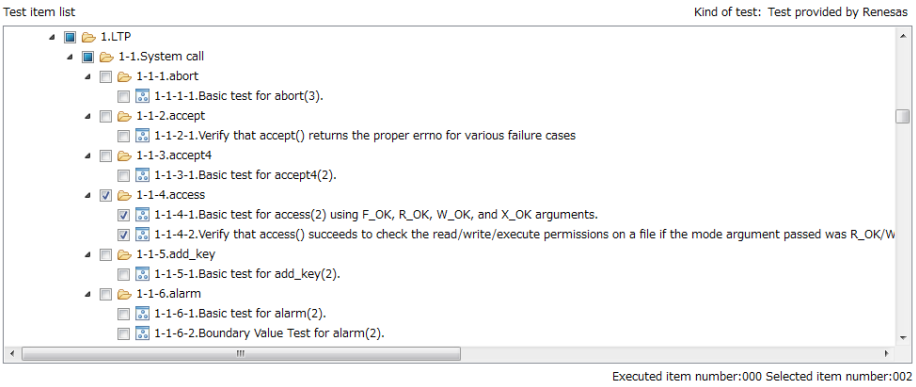


Figure 4-11 After Selecting Items to Be Tested

4.3.2 Connecting the Target Board

Take either of the following steps to connect the target board.

- Select the [Connect] menu item from the [File] menu.
- Click on the [Connect] button on the toolbar.

When the target board has successfully been connected, the [Start] button on the toolbar becomes active and the following message appears in the [Console] section, as shown in Figure 4-12 and Figure 4-13.

```
----------------------Connect Start--------------------------------
root@iwg22m:~#
```

Figure 4-12 Before Connecting Target Board

Figure 4-13 After Connecting Target Board

4.3.3 Starting the Tests

Take either of the following steps to start the tests.

- Select the [Start Tests] menu item from the [Test] menu.
- Click on the [Start Tests] button on the toolbar.

Once the tests start, a log of execution of the tests appears in the [Console] section, as shown in Figure 4-14. The progress bar also appears in the dialog box, as shown in Figure 4-15.

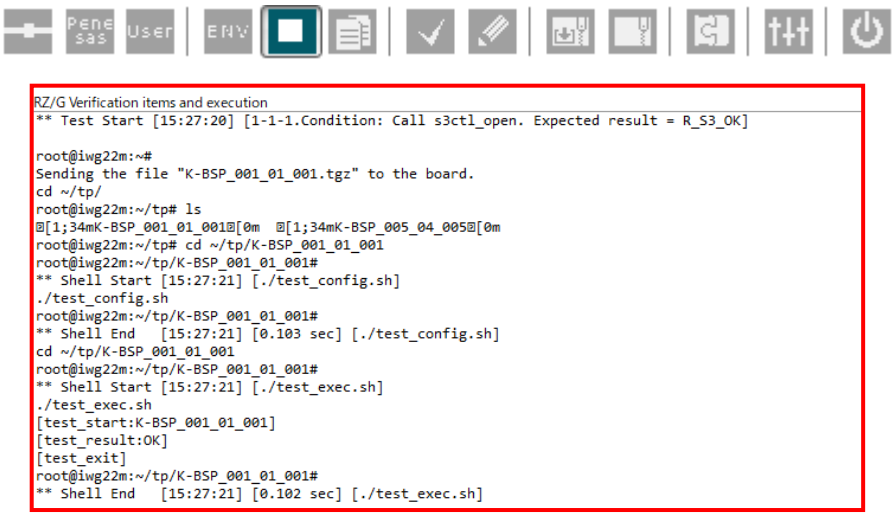


Figure 4-14 Display during Execution of Tests

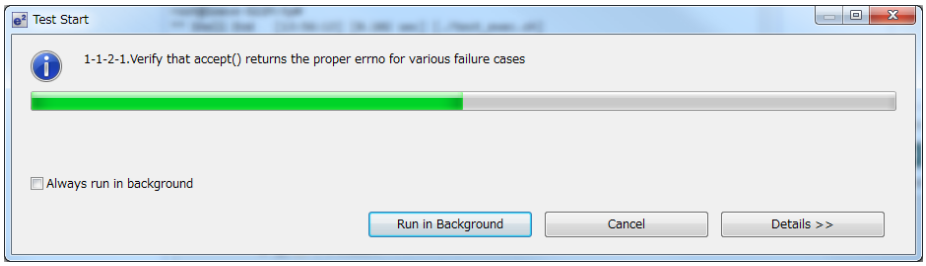


Figure 4-15 Progress Dialog during Execution of Tests

4.3.4 End of Testing and Terminating the Tests

Testing normally ends once selected tests have all been executed. To terminate the tests while they are in progress, take either of the following steps.

- Select the [Terminate test] menu item from the [Test] menu.
- Click on the [Terminate test] button on the toolbar.

When the selected tests have all been executed, an outline of the results appears as shown in Figure 4-16 and the progress dialog box is closed.

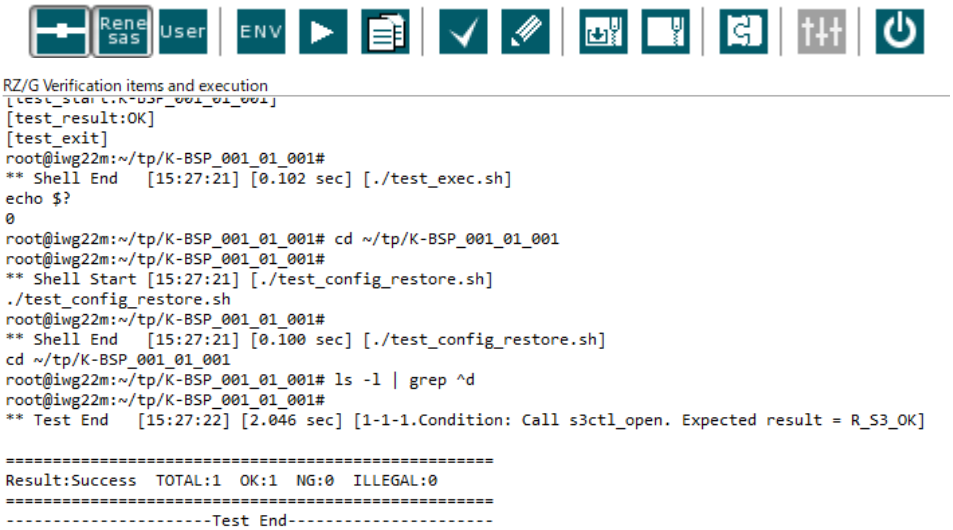


Figure 4-16 Screen when Testing is Completed

Note that even if the test is stopped halfway, the results of the test executed before it is stopped are saved. Refer to 4.4 and confirm the results.

4.3.5 Other Operations from the [Test item list] Section

Clicking the right mouse button on any item in the [Test item list] section displays a menu, as shown in Figure 4-17. Clicking on each item in the displayed menu leads to the respective operations listed in Table 4-3.

Table 4-3 Pop-up Menu in [Test item list] Section

Menu Item	Operation
Put a check in the upper side items from here	Checks checkboxes for all items displayed above the selected item.
Put a check in the lower side items form here	Checks checkboxes for all items displayed below the selected item.
Remove a check in the upper side items from here	Unchecks checkboxes for all items displayed above the selected item.
Remove a check in the lower side items from here	Unchecks checkboxes for all items displayed below the selected item.
Execute items in the lower side from here	Executes tests for all items with checked checkboxes from among the items displayed below the selected one.
Execute items included in this item	Executes the test for the selected item. If the selected item is a group, all tests in the group are executed.

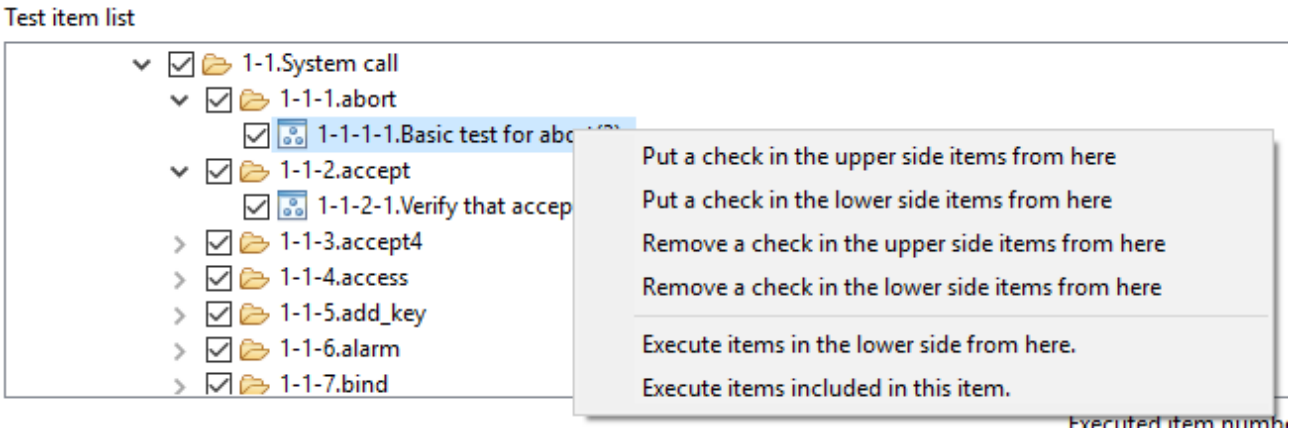


Figure 4-17 Pop-up Menu in [Test item list] Section

4.4 Confirming the Results of Tests

4.4.1 Displaying a List of Test Result Histories

Select [Show View] -> [Other] from the [Window] menu on the e^2 studio. The screen is displayed as shown in Figure 4-18.
On the screen, select [RZ/G Linux Platform tools] -> [RZ/G Verification results]. The window indicating the history of the test results is displayed as shown in Figure 4-19.

The history of the test results, including the target environment, dates when the tests were executed, number of items tested, number of passed items, number of failed items, number of errors found in the tests, and overall results of the tests, is displayed in the [History of test result] section.

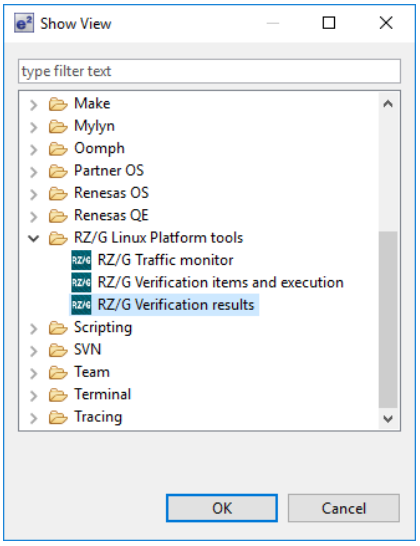


Figure 4-18 Screen for Selecting Window to Display

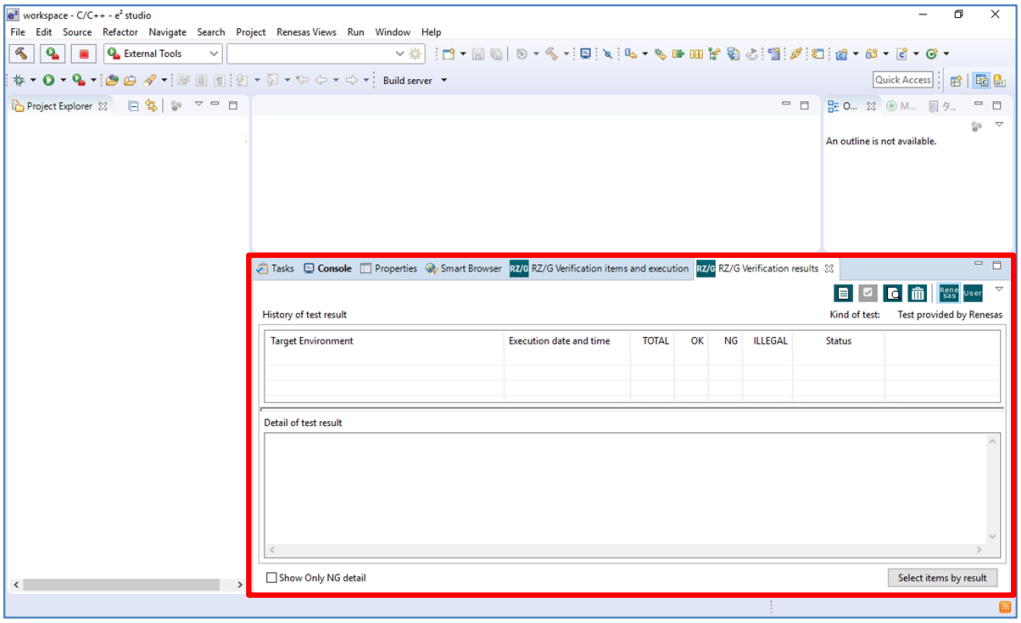


Figure 4-19 [History of test result] Display Window

Select the name of a test result history in the [History of test result] section on the screen, and its detail will be displayed in the [Detail of test result] section as shown in Figure 4-20.
In addition, clicking on the [Show Only NG detail] checkbox selects the exclusive display of details on test results for items for which the result was NG or an error.

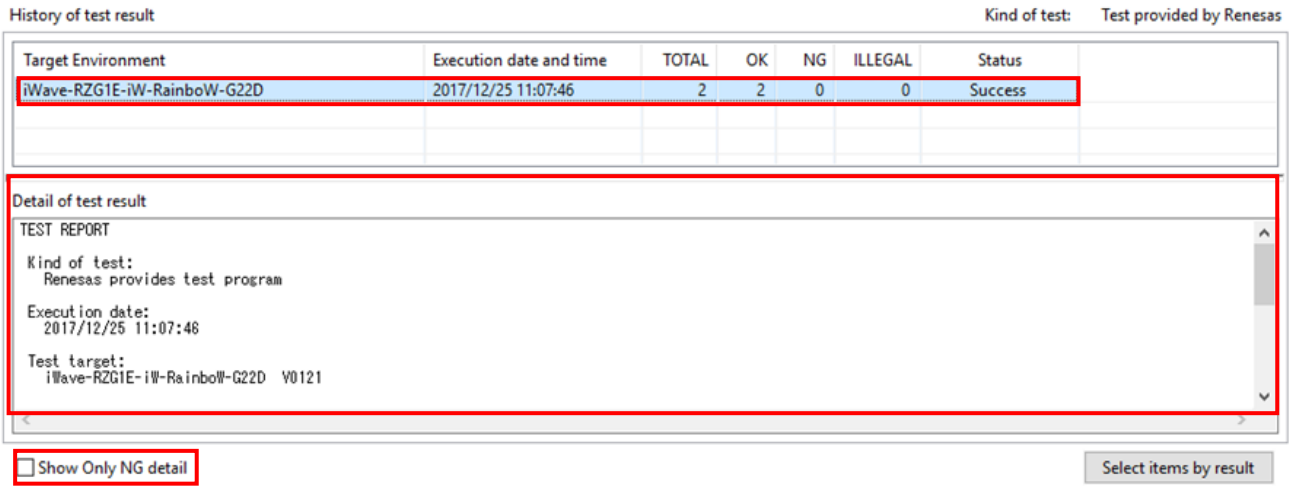


Figure 4-20 Detailed Display of Test Results

4.4.2 Deleting Test Results

To delete the results of executed tests, select the history to be deleted from the [History of test result] section as shown in Figure 4-21, and take either of the following steps.

- Select the [Delete] menu item from the [Result] menu.
- Click on the [Delete] button on the toolbar.

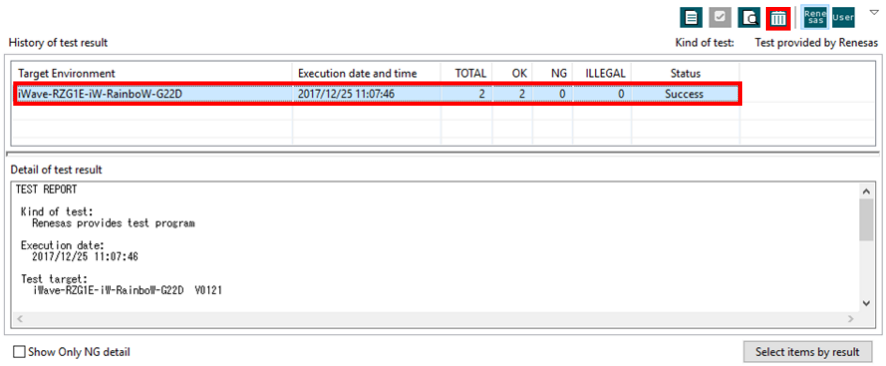


Figure 4-21 Deleting Test Results

After that, the [Delete test result] dialog box shown in Figure 4-22 appears. Clicking on the [OK] button leads to deletion of the selected history and the window becomes as shown in Figure 4-23 in the case of our example.

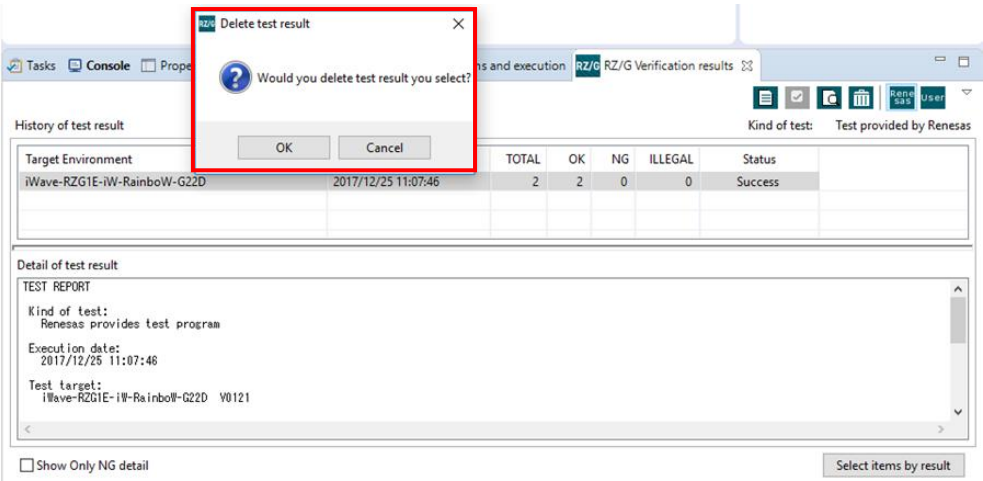


Figure 4-22 Dialog Box for Confirming Whether to Delete Test History

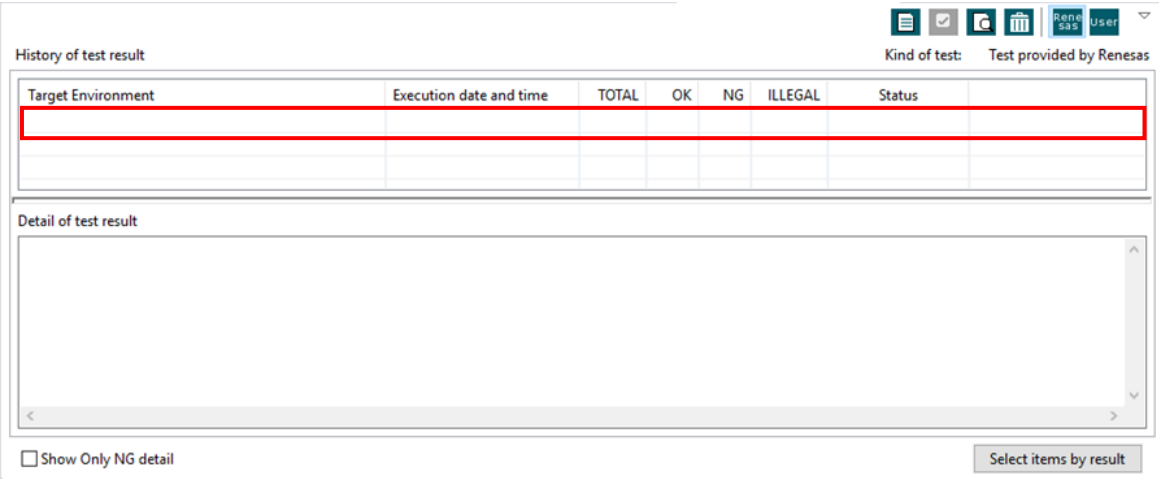


Figure 4-23 Screen after Deleting Test History

4.4.3 Exporting a Report File

To export a report file of the test results, select the history to be exported from the [History of test result] section as shown in Figure 4-24, and take either of the following steps.

- Select [Export] from the [Result] menu and select the [Report Files] menu item.
- Click on the [Report Files] button on the toolbar.

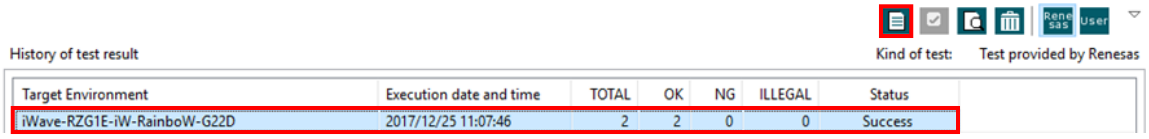


Figure 4-24 Exporting a Report File

After either of the steps described above, the file storage dialog box appears. You can use this dialog box to export the report file to the desired folder.

4.4.4 Exporting the File for Analysis Tool

To save a file for analysis tool, as shown in Figure 4-25, select the history to be exported from the [History of test result] section, and create a file for the analysis tool by taking either of the following steps.
Only the results of the test programs provided by Renesas can be exported as files for the analysis tool.

- Select [Export] from the [Result] menu and select the [File for Analysis tool] menu item.
- Click on the [File for Analysis tool] button on the toolbar.

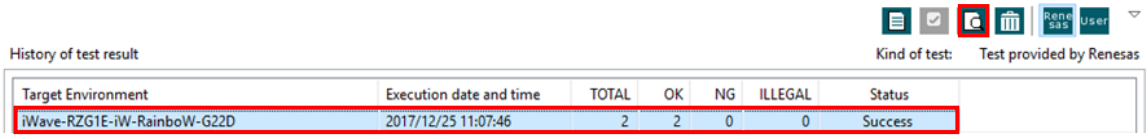


Figure 4-25 Exporting the File for Analysis Tool

After either of the steps described above, the file storage dialog box appears. You can use this dialog box to export the file for the analysis tool to the desired folder

When you change the name of the saved file, be sure to leave the first part of the file name "VerificationResult_" unchanged because changing the first part disables the analysis tool to use the file.

4.4.5 Selecting Test Items from the Results of Tests

This function changes the state of selection of test items shown in the main display by selecting only those items with NG and error results from among the selected tests.
Specifically, the steps involved are as follows.

- Select the history to use in changing the selection from the [History of test result] section, and click on the [Select items by result] button (see Figure 4-26).
- Click on the button for the type of results for tests to be selected in the [Kind of select items] dialog box on the screen (see Figure 4-27).

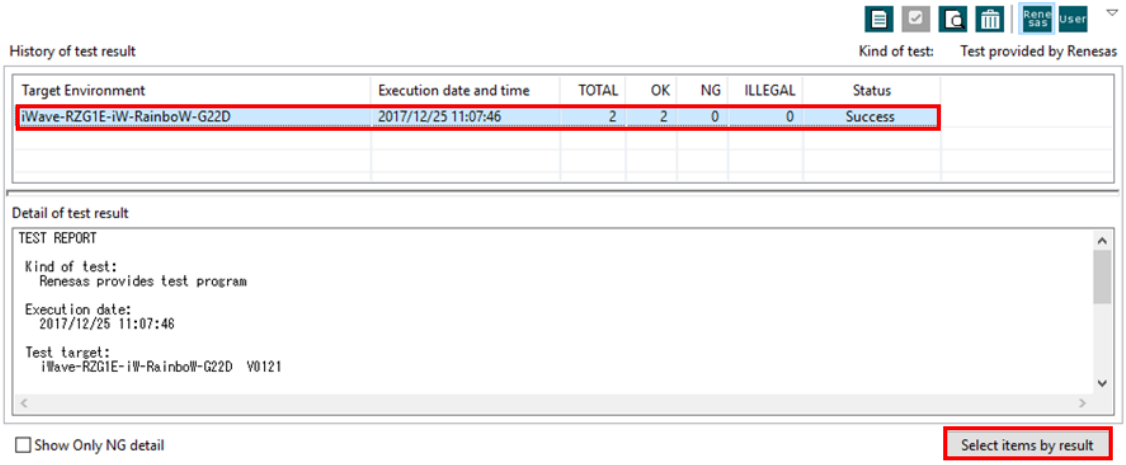


Figure 4-26 Selecting Test Items from the Results

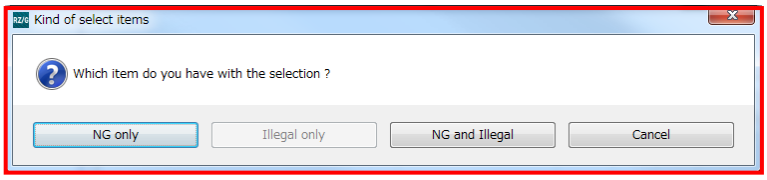


Figure 4-27 [Kind of select items] Dialog Box

After the steps described on the previous page, the test items specified in Figure 4-27 enter the selected state (Figure 4-28 shows a result with [NG only] specified).

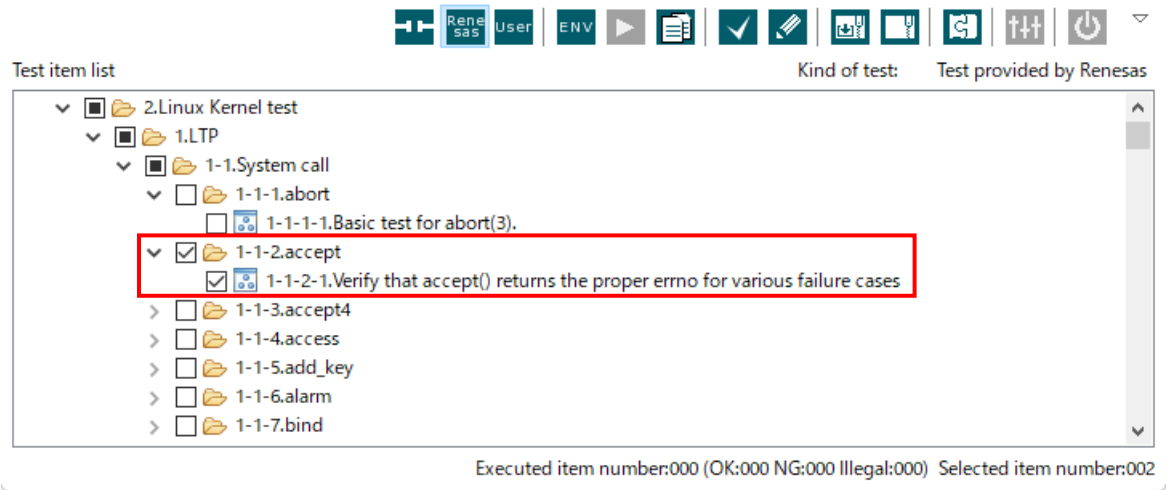


Figure 4-28 Result of Selecting Test Items According to the Results of Tests

4.5 Saving and Restoring the State of Test Item Selection

4.5.1 Saving the State of Test Item Selection

After selecting test items as shown in Figure 4-29, the state of test item selection is saved in either of two ways.

- Select the [Save/Restore data for selection state of test item(s)] menu item from the [Test] menu.
- Click on the [Save/Restore data for selection state of test item(s)] button on the toolbar.

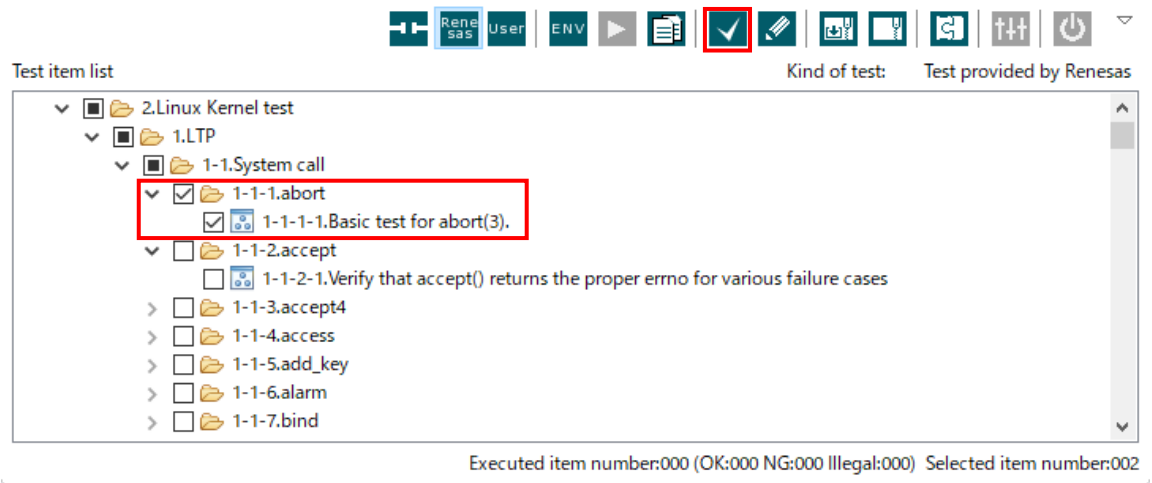


Figure 4-29 Saving the State of Test Item Selection

After doing either of the above, the [Save/Restore data for selection state of test item(s)] dialog box shown in Figure 4-30 appears. Click on the [Save] button.
Next, enter a name for the state data to be saved in the [Save data for selection state of test item(s)] dialog box shown in Figure 4-31 (the name entered is "ABC" in Figure 4-31), and click on the [OK] button.
The state of test item selection is then saved.

After the data are saved, the name specified when saving is displayed in the [Save/Restore data for selection state of test item(s)] dialog box (see Figure 4-32).

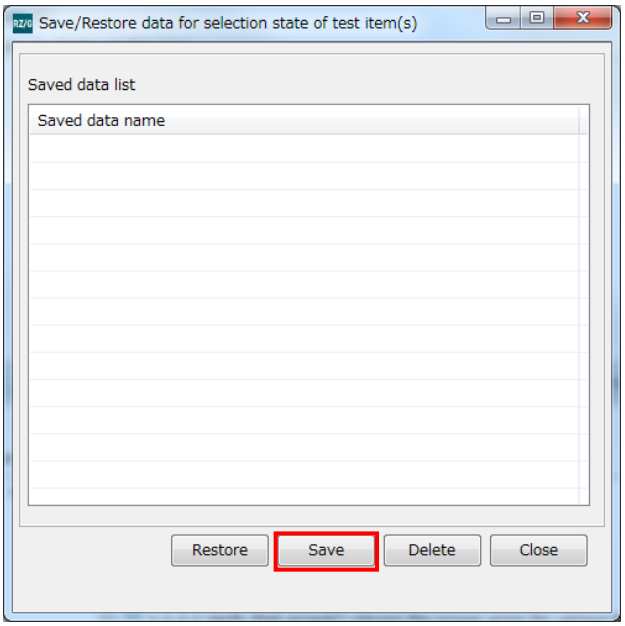


Figure 4-30 [Save/Restore data for selection state of test item(s)] Dialog Box

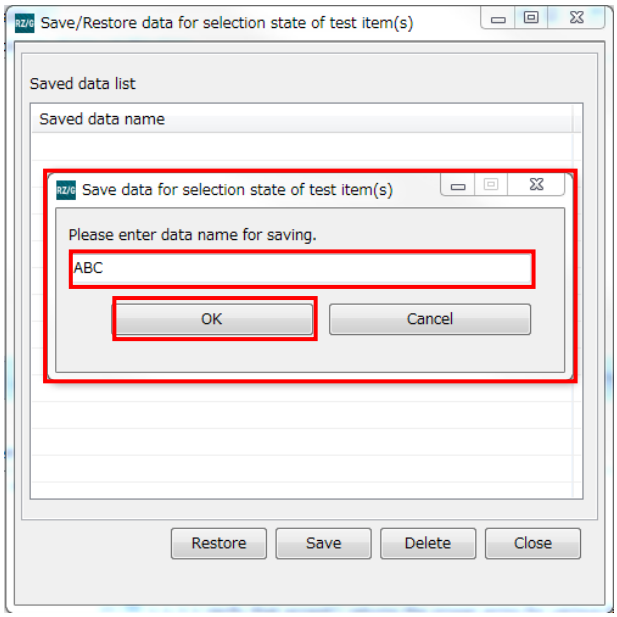


Figure 4-31 [Save data for selection state of test item(s)] Dialog Box

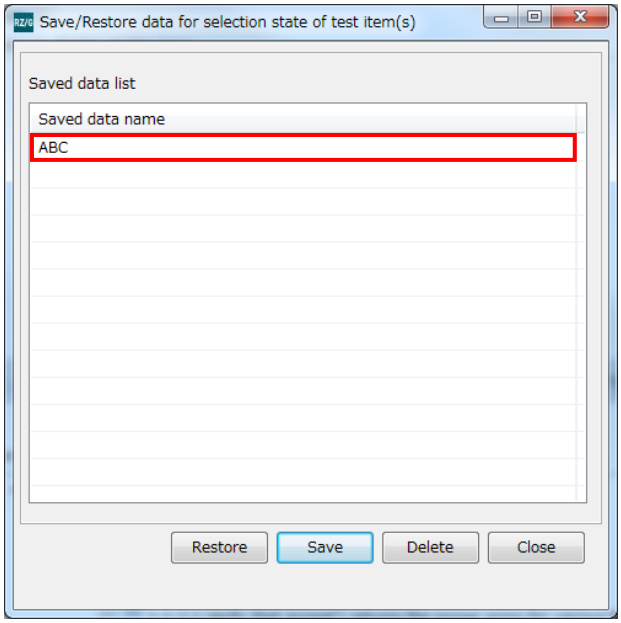


Figure 4-32 Result of Saving the State of Test Item Selection

4.5.2 Restoring the State of Test Item Selection

Restore data on a state of test item selection in either of the following ways.

- Select the [Save/Restore data for selection state of test item(s)] menu item from the [Test] menu.
- Click on the [Save/Restore data for selection state of test item(s)] button on the toolbar (see Figure 4-33).

Figure 4-33 Saving or Restoring the State of Test Item Selection

The [Save/Restore data for selection state of test item(s)] dialog box shown in Figure 4-34 appears. Select an item to be restored from the [Saved data list] section and click on the [Restore] button. The state of test item selection is then restored, as shown in Figure 4-35.

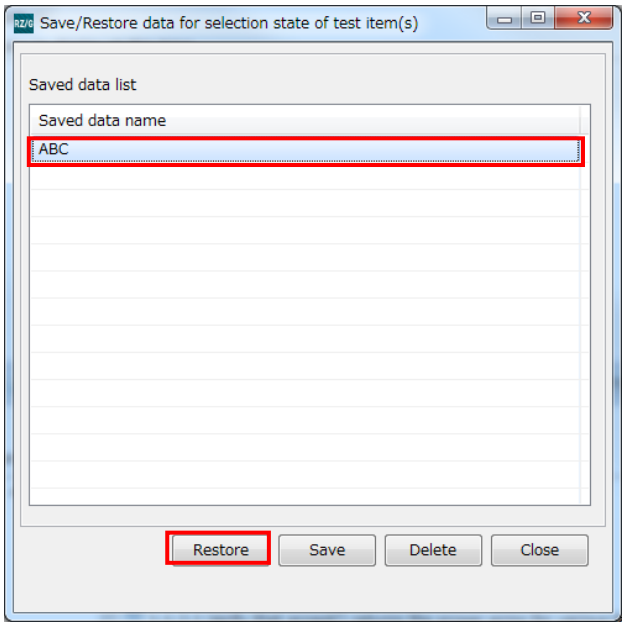


Figure 4-34 [Save/Restore data for selection state of test item(s)] Dialog Box

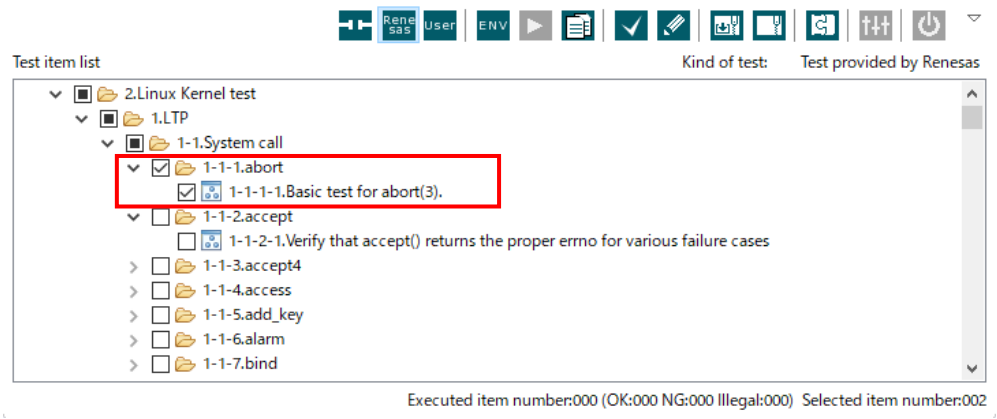


Figure 4-35 Result of Restoring the State of Test Item Selection

4.5.3 Deleting a State of Test Item Selection

Delete data on a state of test item selection in either of the following ways.

- Select the [Save/Restore data for selection state of test item(s)] menu item from the [Test] menu.
- Click on the [Save/Restore data for selection state of test item(s)] button on the toolbar (see Figure 4-36).

Figure 4-36 Saving or Restoring the State of Test Item Selection

After doing either of the above, the [Save/Restore data for selection state of test item(s)] dialog box shown in Figure 4-37 appears. Select an item to be deleted from the [Saved data list] section and click on the [Delete] button. The [Delete data for selection state of test item(s)] dialog box shown in Figure 4-38 then appears, requesting confirmation of whether or not you wish to delete the selected data. Click on the [OK] button. As a result, the data on the state of test item selection are deleted, and the item is deleted from the [Saved data list] section, as shown in Figure 4-39.

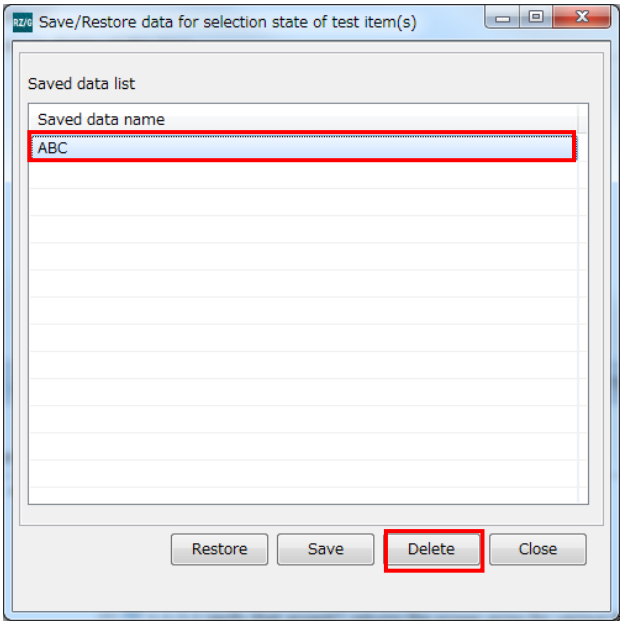


Figure 4-37 [Save/Restore data for selection state of test item(s)] Dialog Box

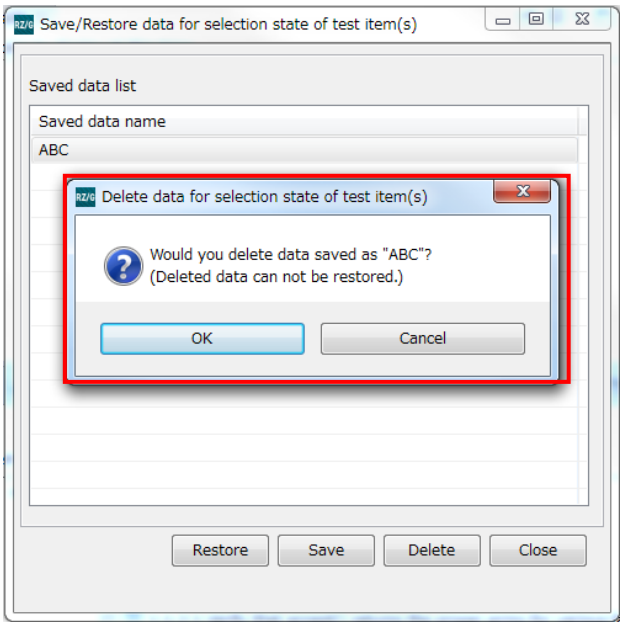


Figure 4-38 [Delete data for selection state of test item(s)] Dialog Box

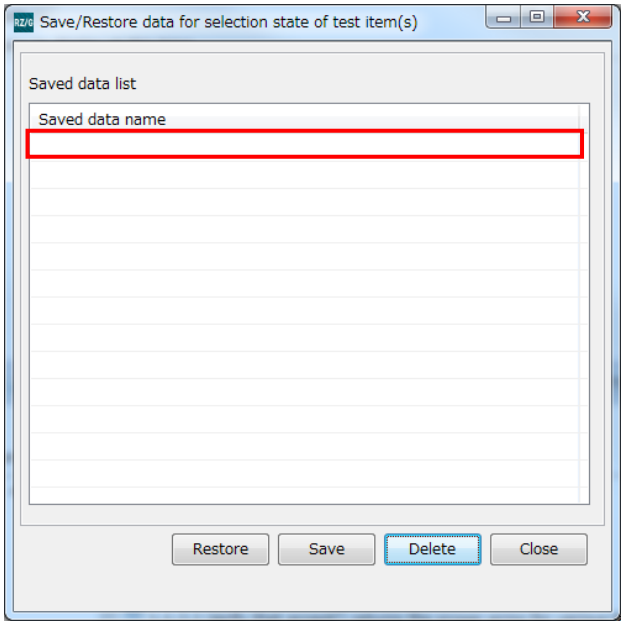


Figure 4-39 Result of Deleting a State of Test Item Selection

4.6 Registering and Editing Test Programs Created by Users

Test programs created by users can be registered with and executed by the software verification tool. How to register test programs and how to edit them after registration are described on the following pages.

Note that the test programs must be created in accord with the separate Guide for Creating Test Programs for the Software Verification Tool.

4.6.1 Registering Test Programs in a Single Round

To register a test program, take either of the following steps to open the [Register user test programs] window as shown in Figure 4-40.

- Select the [Register user test programs] menu item from the [Test] menu.
- Click on the [Register user test programs] button on the toolbar.

Figure 4-40 Registering User-Created Test Programs

After either of the steps described above, the [Register user test programs] dialog box shown in Figure 4-41 appears.

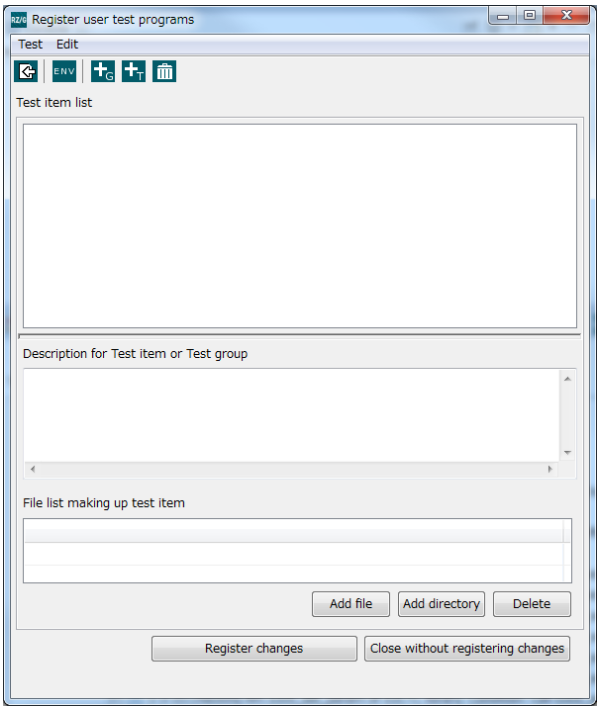


Figure 4-41 [Register user test programs] Window

Taking either of the following steps in this window as shown in Figure 4-42 displays the window for use in selecting a folder shown in Figure 4-43.

- Select the [Import user test programs] menu item from the [Test] menu.
- Click on the [Import user test programs] button on the toolbar.

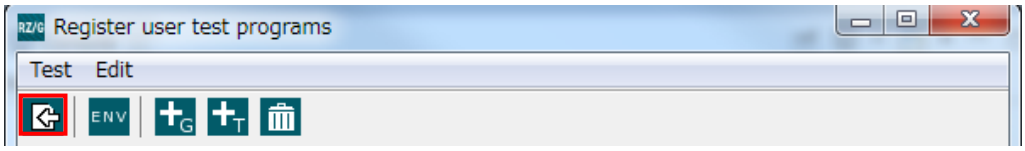


Figure 4-42 Registering User-Created Test Programs in a Single Round

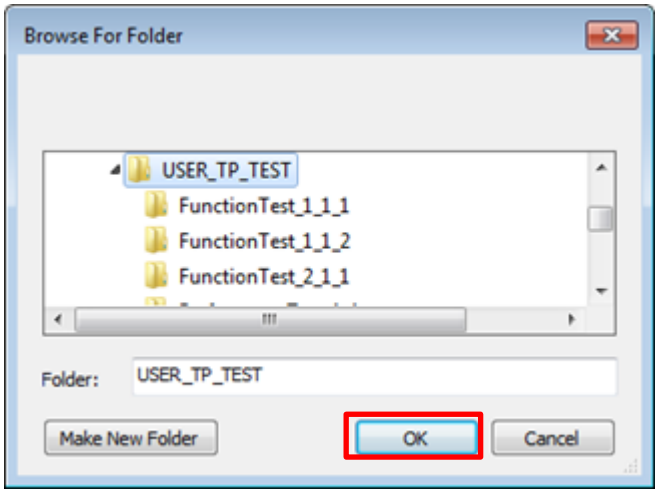


Figure 4-43 Folder Selection Dialog Box

Specifying the folder to which the test programs have been loaded on the folder selection screen shown in Figure 4-43 and clicking on the [OK] button open the [Import user test programs] dialog box shown in Figure 4-44. Enter the name and version number of the environment to be added on this screen, and click on the [OK] button. (For the name of the environment, the form of halfwidth "alphabet + alphabet/numeral/hyphen" can be specified.)

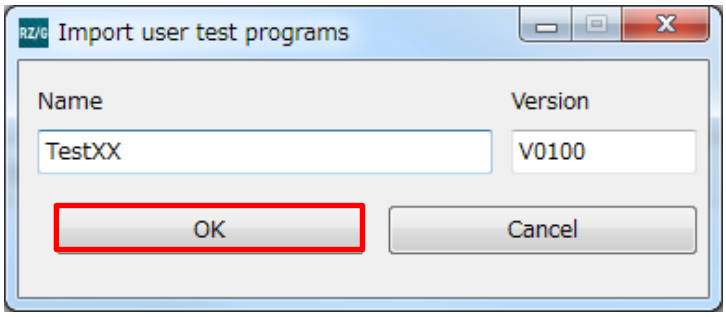


Figure 4-44 [Import user test programs] Dialog Box

On completion of the procedure described on the previous pages, the set of loaded test programs will have been stored immediately under the specified folder. Figure 4-45 shows the window after registration.
Since registration in a single round has been completed at this time, [Register user test programs] dialog box can be closed by clicking on either [Register changes] or [Close without registering changes] button.

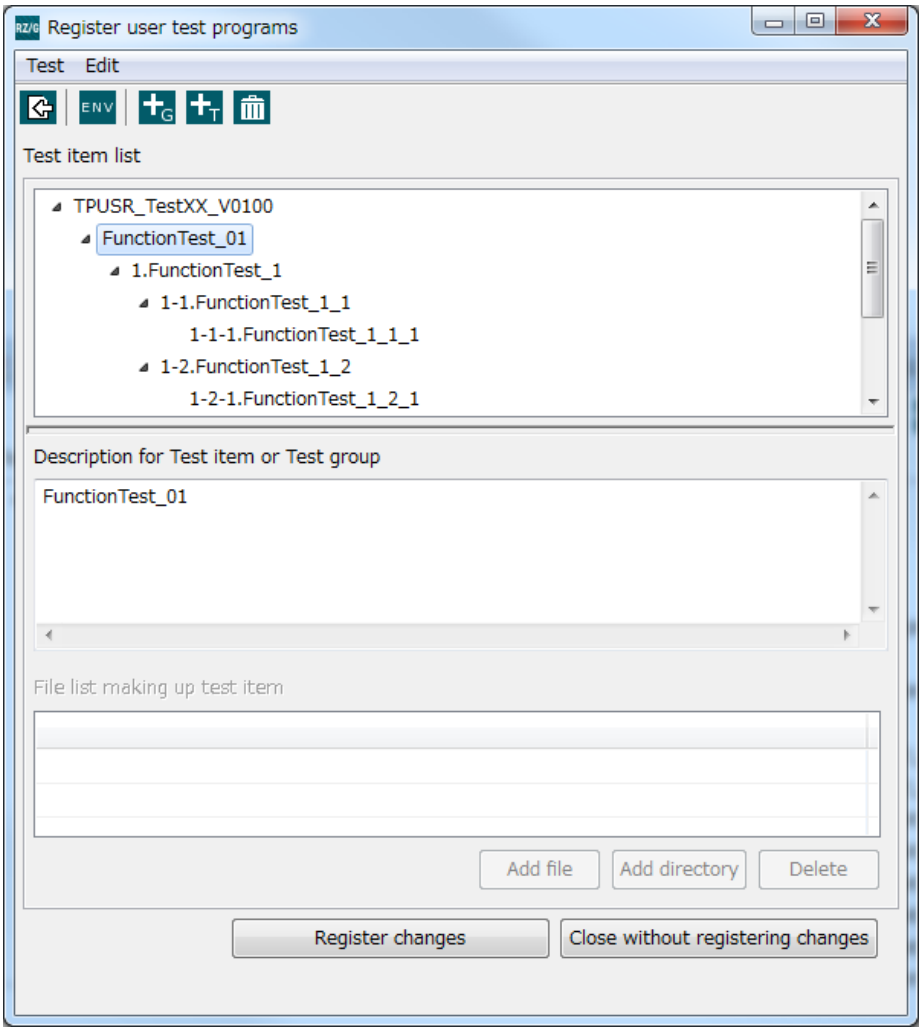


Figure 4-45 Screen after Registration in a Single Round of Test Programs Created by a User

4.6.2 Editing Configuration of Test Programs

4.6.2.1 Selections for the Test Environments

To edit the configuration of the test programs, first open the [Register user test programs] window according to the procedure described in 4.6.1. Then take either of the following steps to select the test environment to be edited.

(1) Select the [Select target Environment] menu item from the [Edit] menu, or click on the [Select target Environment] button (the [ENV] label) on the toolbar.

The list of the environments for the test programs in [Folder to which test programs created by users are to be loaded] is displayed in a drop-down list from the [Test developed by users] text box in the dialog box shown in Figure 4-46. Select the suitable environment.

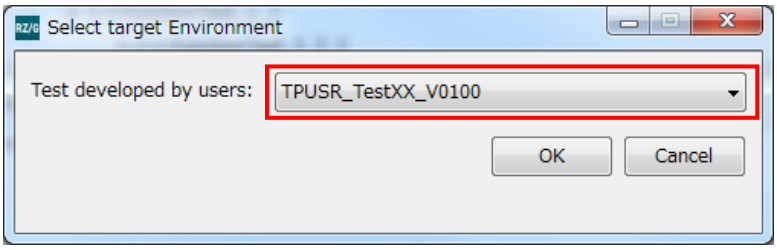


Figure 4-46 Selections for the Test Environments

4.6.2.2 Adding an Item to Be Tested

To add an item to be tested to a registered test environment, take the following action.

When adding is completed, proceed to 4.6.2.3 Adding a Test Program, or click on the [Register changes] button in the [Register user test programs] window to save the added contents.

1. Select the node to which the new test item is to be added, and click on the [Add test group] button on the toolbar as shown in Figure 4-47.

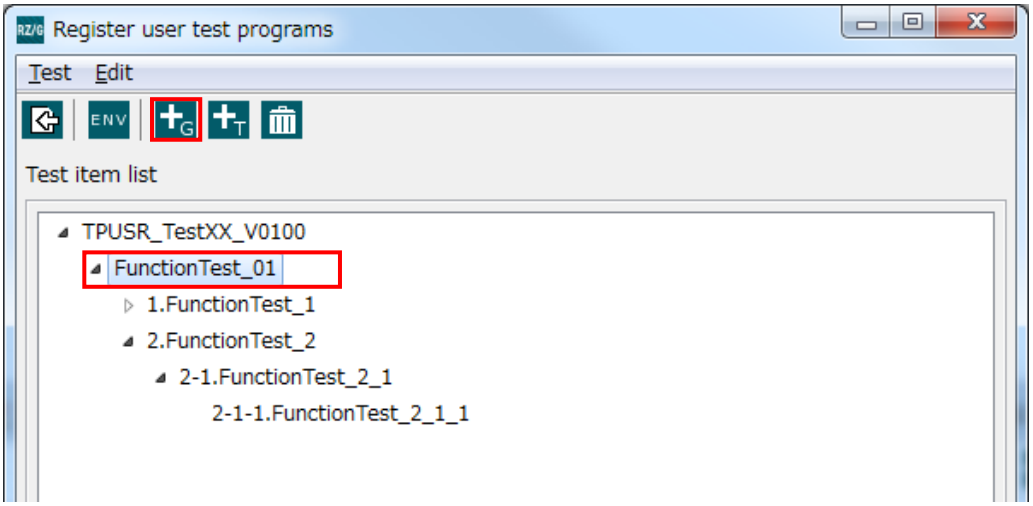


Figure 4-47 Adding an Item to Be Tested

2. Once the [Add test group] dialog box shown in Figure 4-48 appears, enter the number of the group to be added and click on the [OK] button
   The entered value is used as the test item number.

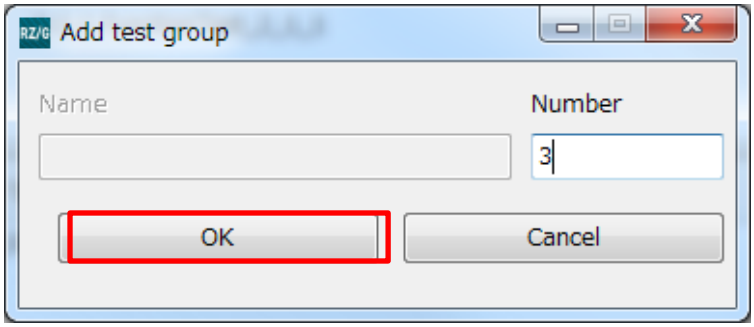


Figure 4-48 [Add test group] Dialog Box

3. After the steps described on the previous page, the added function test is displayed in the [Test item list] section. As shown in Figure 4-49, select The added test and edit the name of the group in the [Description for Test item or Test group] text box.

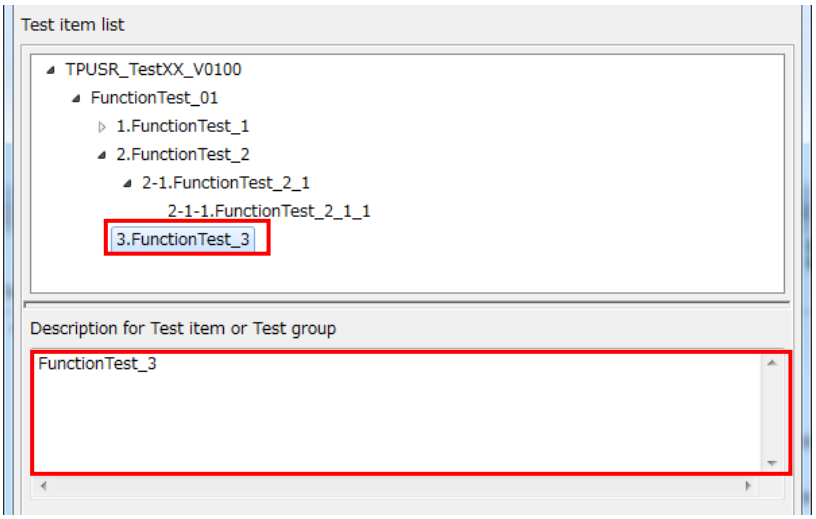


Figure 4-49 Editing the Test Group Name

4.6.2.3 Adding a Test Program

To add a test program to a registered test environment, take the following action.

When adding is completed, proceed to 4.6.2.2 Adding an Item to Be Tested, or click on the [Register changes] button in the [Register user test programs] window to save the added contents.

1. Select the group to which the new test program is to be added, and click on the [Add test item] button on the toolbar.

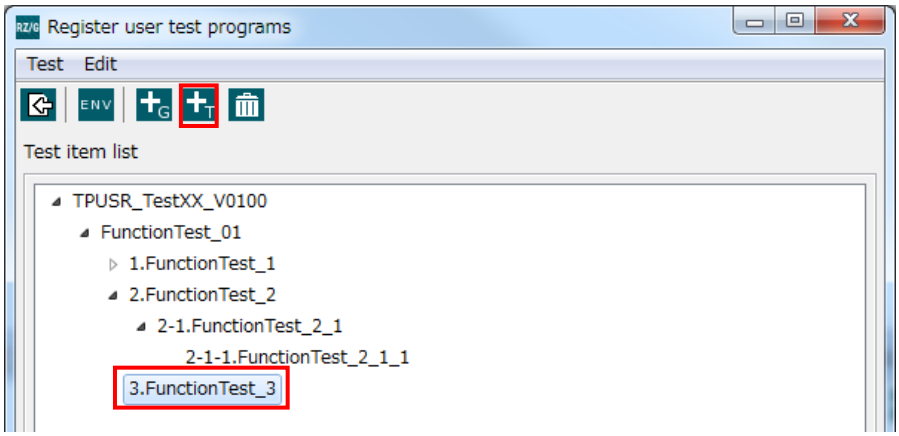


Figure 4-50 Selecting Where to Add a Test Program

2. Once the [Add test item] dialog box appears, enter a value in the [Number] text box and click on the [OK] button.
   The entered value is used as the test item number.

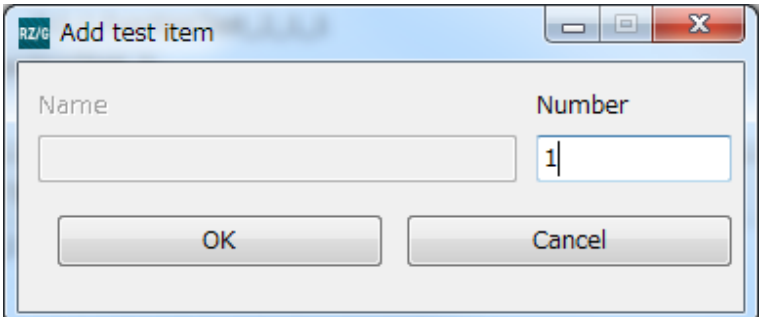


Figure 4-51 Selecting Where to Add a Test Program

3. The added test is displayed in the [Test item list] section. Select the added function test and edit it, as shown in Figure 4-52.

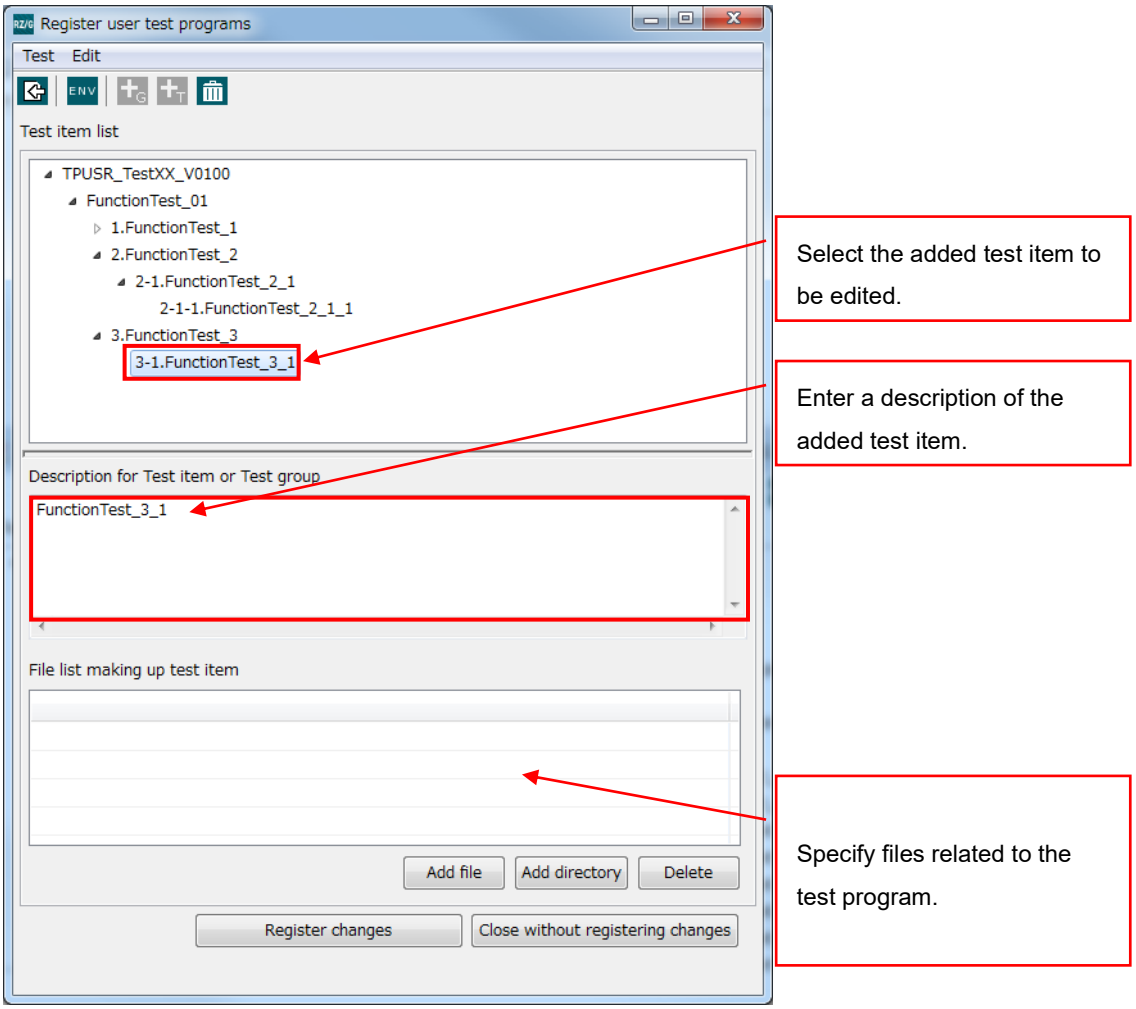


Figure 4-52 Editing a Test Item

4.6.2.4 Deleting a Test Group or Test Item

Clicking on the [Delete] button on the toolbar, indicated in Figure 4-53, deletes a selected test group or test item.

Figure 4-53 Deleting a Test Group or Test Item

Note: Once a test group or test item has been deleted, it cannot be restored.

4.7 Importing or Deleting the Test Programs for the Software Add-on

If you have purchased the software add-on provided with the test programs, you can use the procedure described on the following pages to import and execute the test programs with the verification tool.

4.7.1 Importing the Test Program for Software Add-on

Take either of the following steps to specify the zip file provided with the software add-on and import the test programs.

- Select the [Management of Add-on TPs] menu item from the [Test Program] menu.
- Click on the [Management of Add-on TPs] button on the toolbar.

Click on the [Import] button in the [Management of Add-on TPs] window as shown in Figure 4-54. The file selection dialog box shown in Figure 4-55 will open. Use the dialog box to select the zip file for the test program to be imported. After that, close the [Management of Add-on TPs] window, and the test program will be displayed in the [Test item list] section.

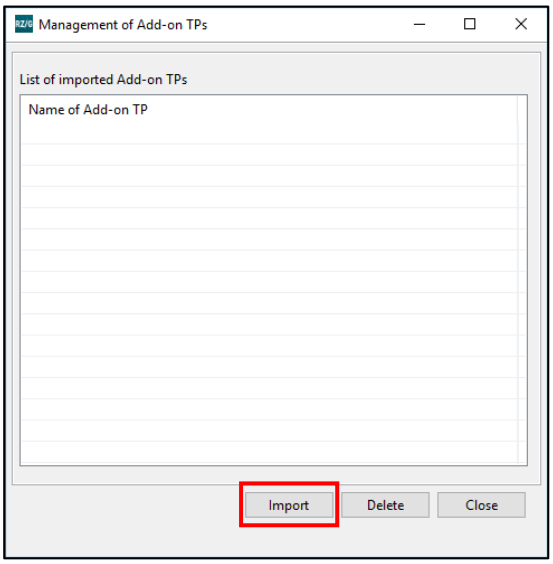


Figure 4-54 [Management of Add-on TPs] Window

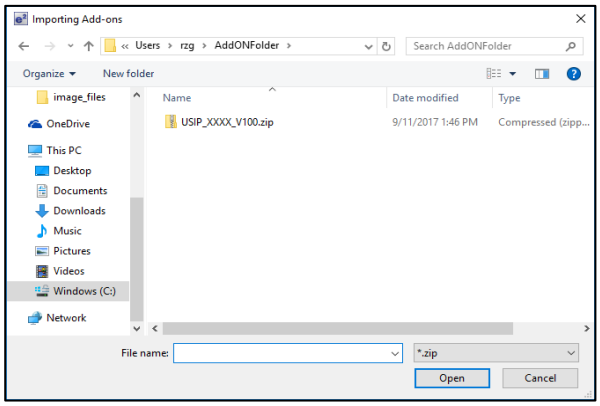


Figure 4-55 File Selection Dialog Box

4.7.2 Deleting the Test Program for Software Add-on

Take either of the following steps to delete an imported test program for the software add-on.

- Select the [Management of Add-on TPs] menu item from the [Test Program] menu.
- Click on the [Management of Add-on TPs] button on the toolbar.

Select the test program to be deleted in the [Management of Add-on TPs] window as shown in Figure 4-56 and click on the [Delete] button.
After that, close the [Management of Add-on TPs] window, and the test program will not be displayed in the [Test item list] section.

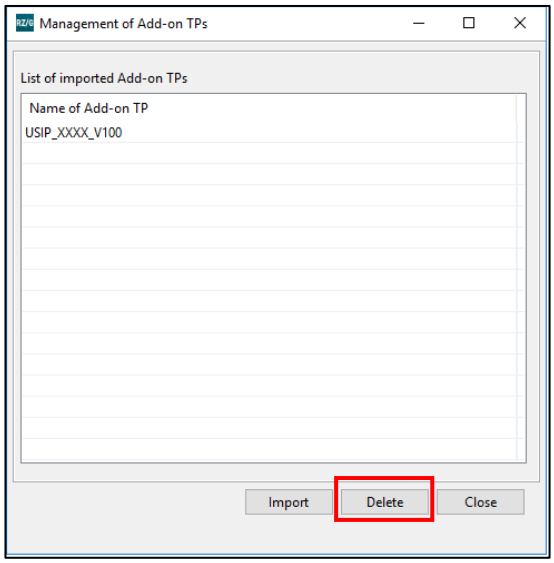


Figure 4-56 [Management of Add-on TPs] Window

4.8 Registering a Set of Test Programs as an Embedded Linux Project

To register a set of test programs as an embedded Linux project, select the test programs to be registered from the [Test item list] section and proceed with either of the following steps.

- Select the [Registering a set of TPs to a build project] menu item from the [Test Program] menu.
- Click on the [Registering a set of TPs to a build project] button on the toolbar.

Select the project for registering the test programs and click on the [OK] button in [Select Embedded Linux Project] window shown in Figure 4-57. This registers the set of test programs as the embedded Linux project as shown in Figure 4-58.

After registering the programs, use either of the following methods to load the test programs to the target board: build the embedded Linux project and create a Linux boot image (see section 3.8) or write them as user programs (see section 3.10.3).

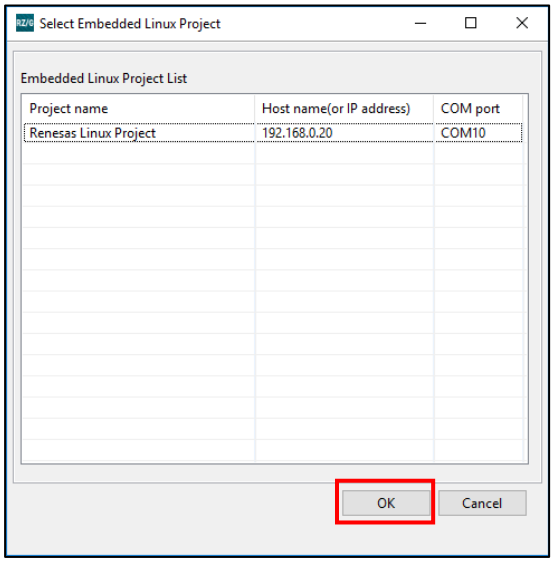


Figure 4-57 [Select Embedded Linux Project] Window

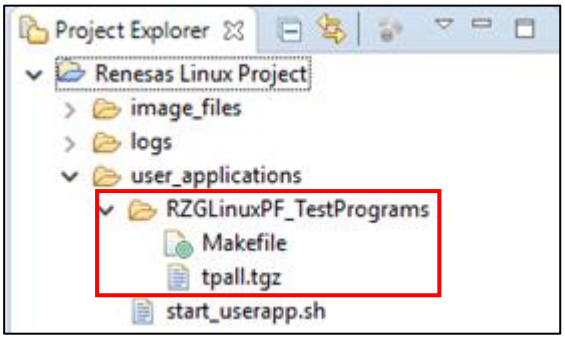


Figure 4-58 Result of Registering a Set of Test Programs as an Embedded Linux Project

Note that the test programs registered by using this function are to be stored in the directory shown below on the target board. Refer to section 4.2.3 to specify the destination directory for storage of the test programs.

Destination directory for storage of the test programs: /home/root/prestored_tp_set

4.9 Saving a Set of Test Programs

To store a set of test programs in your desired folder, select the test programs to be stored from the [Test item list] section and proceed with either of the following steps.

- Select the [Saving a file containing a set of TPs] menu item from the [Test Program] menu.
- Click on the [Saving a file containing a set of TPs] button on the toolbar.

The dialog box for storing a file shown in Figure 4-59 will open. Use this dialog box to store the set of test programs in your desired location.

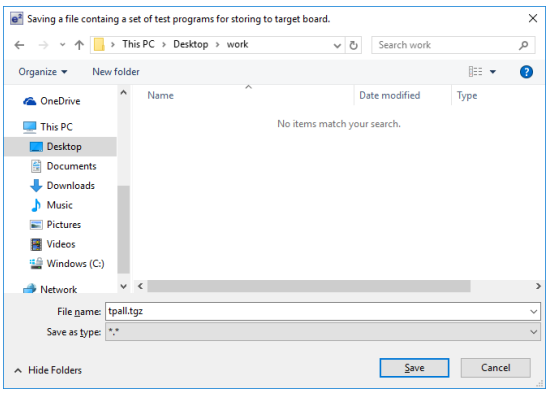


Figure 4-59 Dialog Box for Storing a File

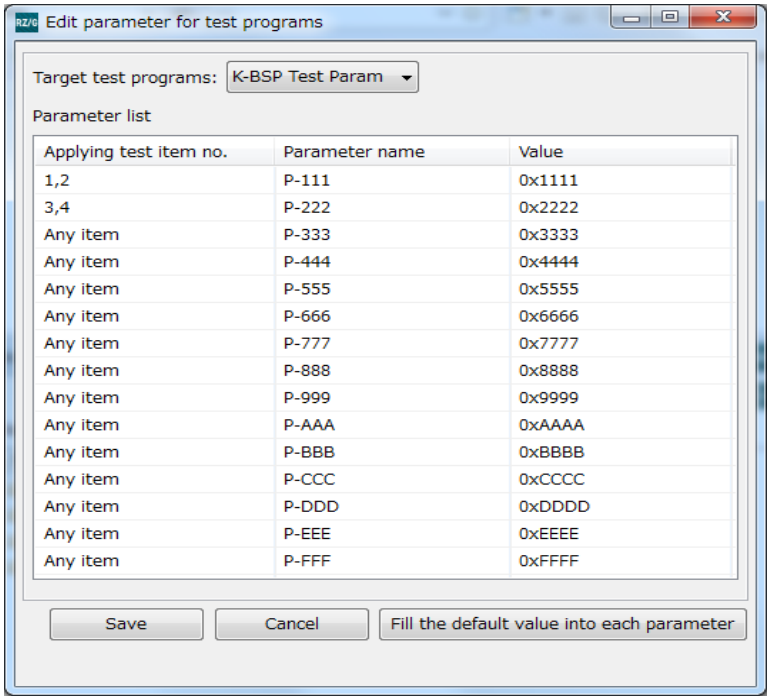
4.10 Editing a parameter for Test Programs

When the execution of the test program, parameters (such as device ID) according to the user environment are set.
Proceed with the following steps when editing this parameter.

- Select the [Edit parameter for test programs] menu item from the [Test Program] menu.
- Click on the [Edit parameter for test programs] button on the toolbar.

Select the TP included test items you want to edit from the [Target test programs] as shown in Figure 4-60, and the list of parameters used by the selected TP is displayed. After edit the parameter click on the [Save] button. After this, this TP is executed using the set parameter.
When returning each parameter to the default state, Click the [Fill the default value into each parameter] button and click the [Save] button.

,



Applying test item no.	Parameter name	Value
1,2	P-111	0x1111
3,4	P-222	0x2222
Any item	P-333	0x3333
Any item	P-444	0x4444
Any item	P-555	0x5555
Any item	P-666	0x6666
Any item	P-777	0x7777
Any item	P-888	0x8888
Any item	P-999	0x9999
Any item	P-AAA	0xAAAA
Any item	P-BBB	0xBBBB
Any item	P-CCC	0xCCCC
Any item	P-DDD	0xDDDD
Any item	P-EEE	0xEEEE
Any item	P-FFF	0xFFFF

Figure 4-60 Editing a parameter for test program Window

5. Handling the Analysis Tool

The Analysis tool can be used only when purchased.

The Analysis tool displays hints to assist in solving problems according to the results of the verification tool to test the items provided by Renesas.

Section 5.1 describes how to setup environment for using the development environment service web server (In this subsection, that is described as the "web server"). and section 5.2 describes how to handle the Analysis Tool.

5.1 Settings to the web server

The following describes how to handle the web server.

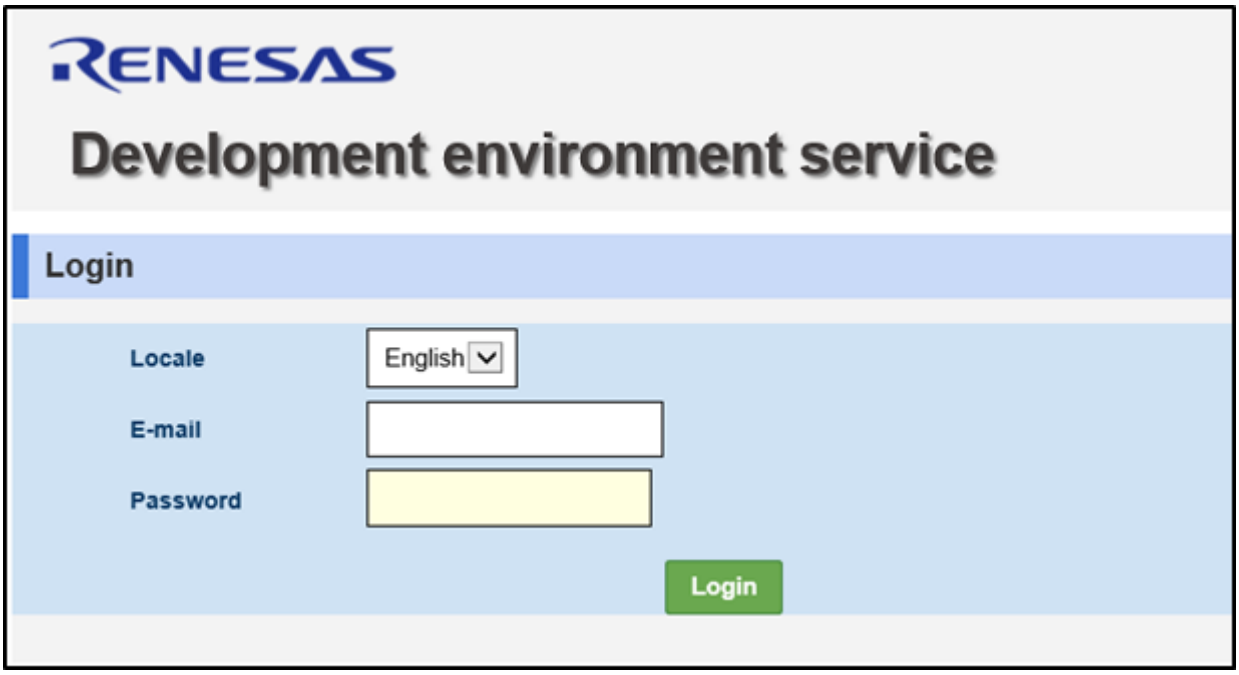
5.1.1 Outline of Functions

The web server can control the following functions.

Function Name	Description
Viewing the information	Displays the information from the web server.
Deleting the information	Deletes the information from the web server.
Changing user information	Changes user information (user ID, password, and language).
Downloading	Downloads required files such as documents.

5.1.2 Logging in

Go to the URL of which you have been notified by Renesas. You will see the login page as shown below.



When you log in to the web server for the first time, you must enter the following information.

- The email address you have received from Renesas
- The password you have received from Renesas

5.1.3 Description of Functions

5.1.3.1 Viewing the information

This subsection describes how to view the information conveyed from the web server.

1. Once you are logged in to the web server, a list of subjects of the information is displayed in the [Top page] window.
2. Click on the title with the information you wish to view.

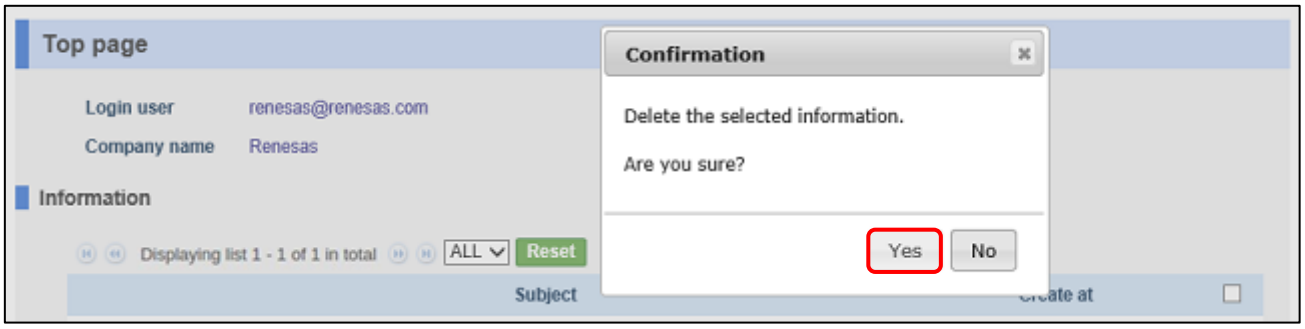
3. Click on the [List] button to return from the [Information detail] window to the [Top page] window.

5.1.3.2 Deleting the information

This subsection describes how to delete the information.

1. Check the checkbox to the right of the information you wish to delete from the list, and then click on the [Delete checked subjects and information.] button.

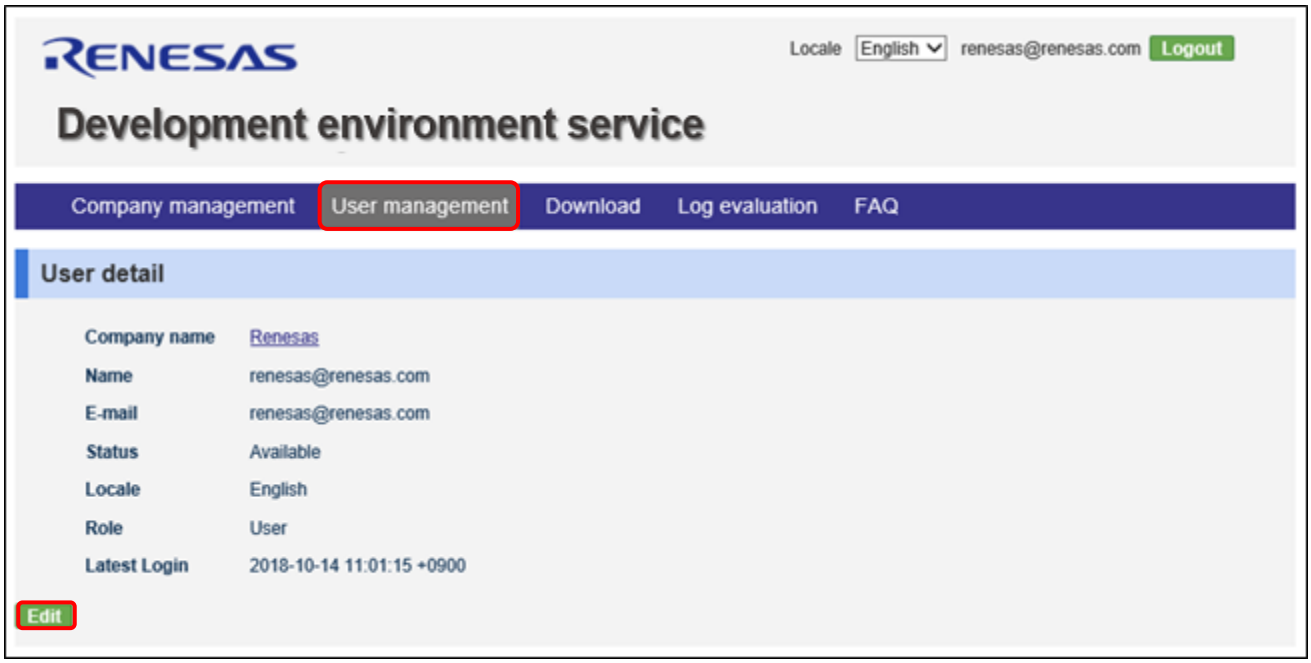
2. Click on [Yes] in the [Confirmation] dialog box.



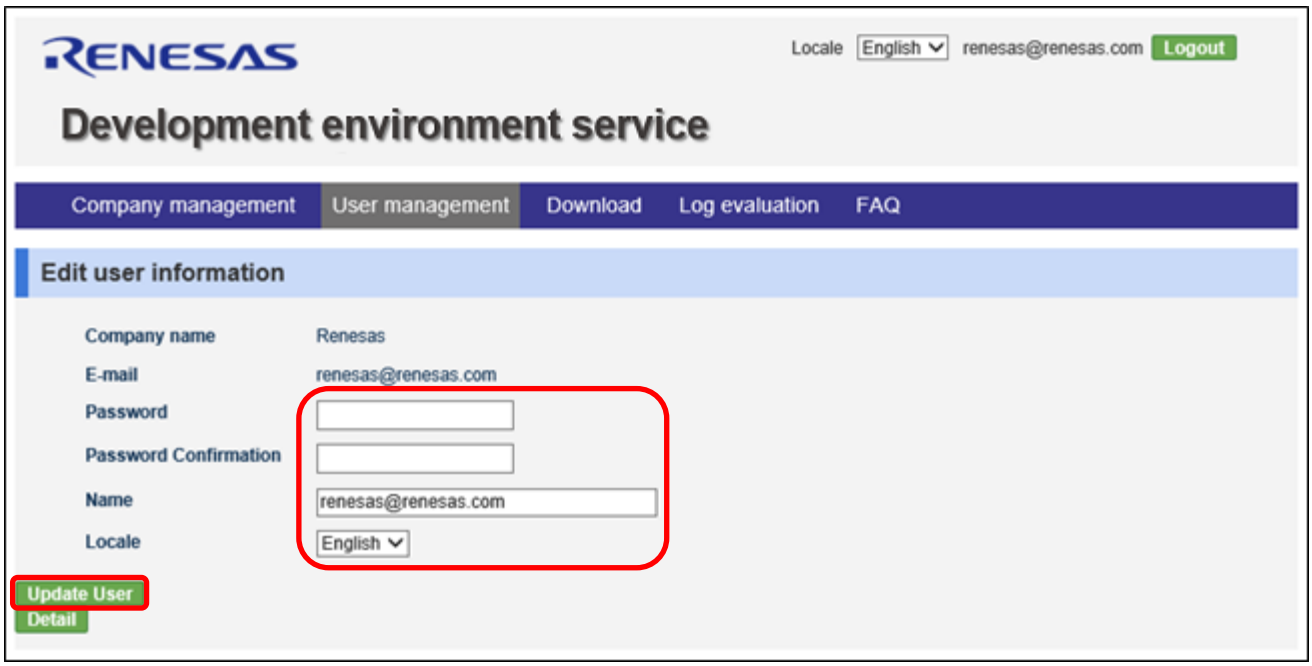
5.1.3.3 Changing User Information

This subsection describes how to change the user information of the registered user.

1. Once you are logged in to the web server, select the [User management] menu to display the [User detail] window, and click on the [Edit] button.



2. In the [Edit user information] window, enter the parameters to be changed. Once you have entered the required information, click on the [Update User] button.

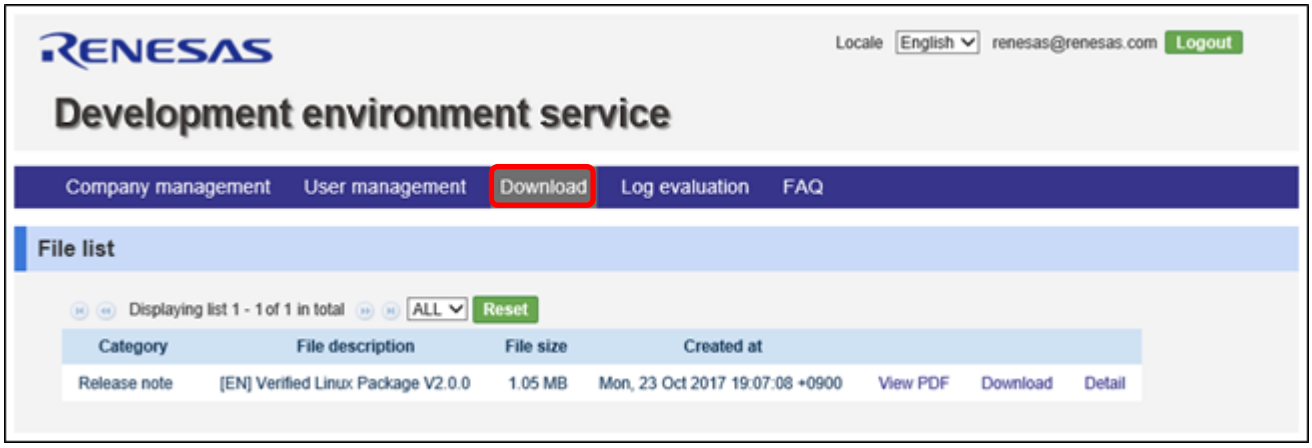


Parameter	Description
Password	Used for logging in to the web server. For details, see 5.1.2 Logging in. Note: When you do not change the password, click on the [Update User] button with the text box left blank.
Password Confirmation	Re-enter the new password.
Name	Enter the same content as "E-mail" field.
Locale	Select either [English or Japanese] as the language used for the web server.

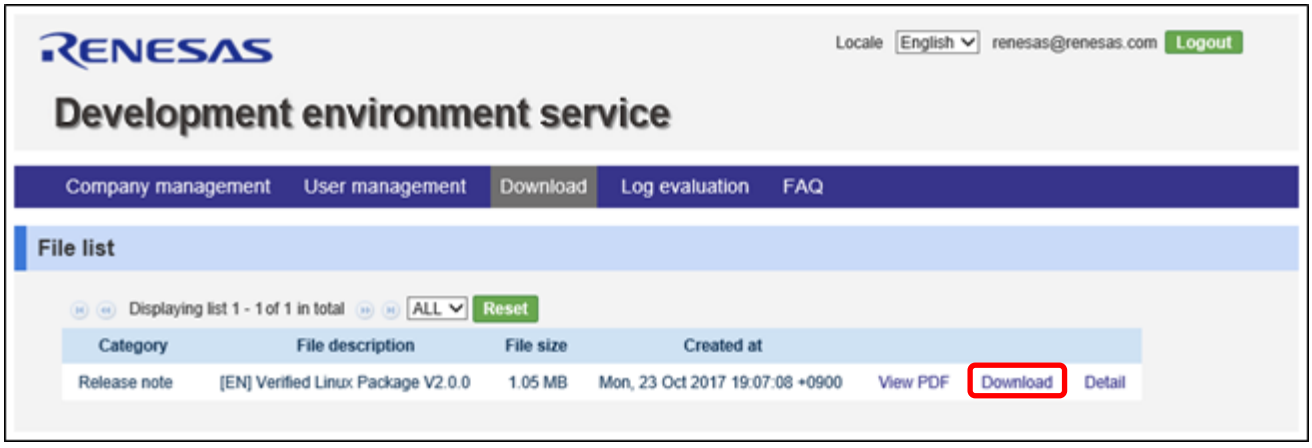
5.1.3.4 Downloading

This subsection describes the procedure for downloading required files such as manuals.

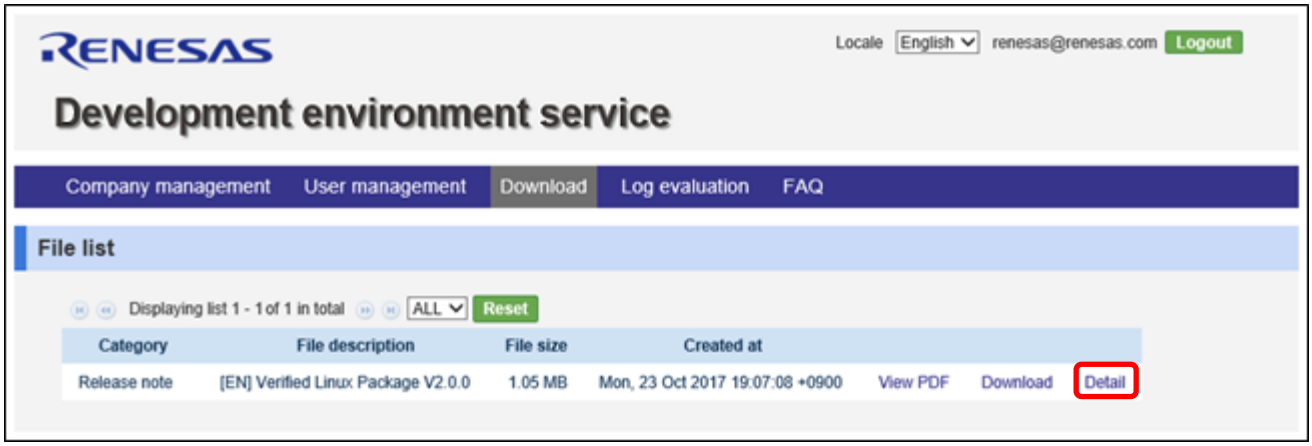
1. Once you are logged in to the web server, select the [Download] menu item to display the [File list] window.



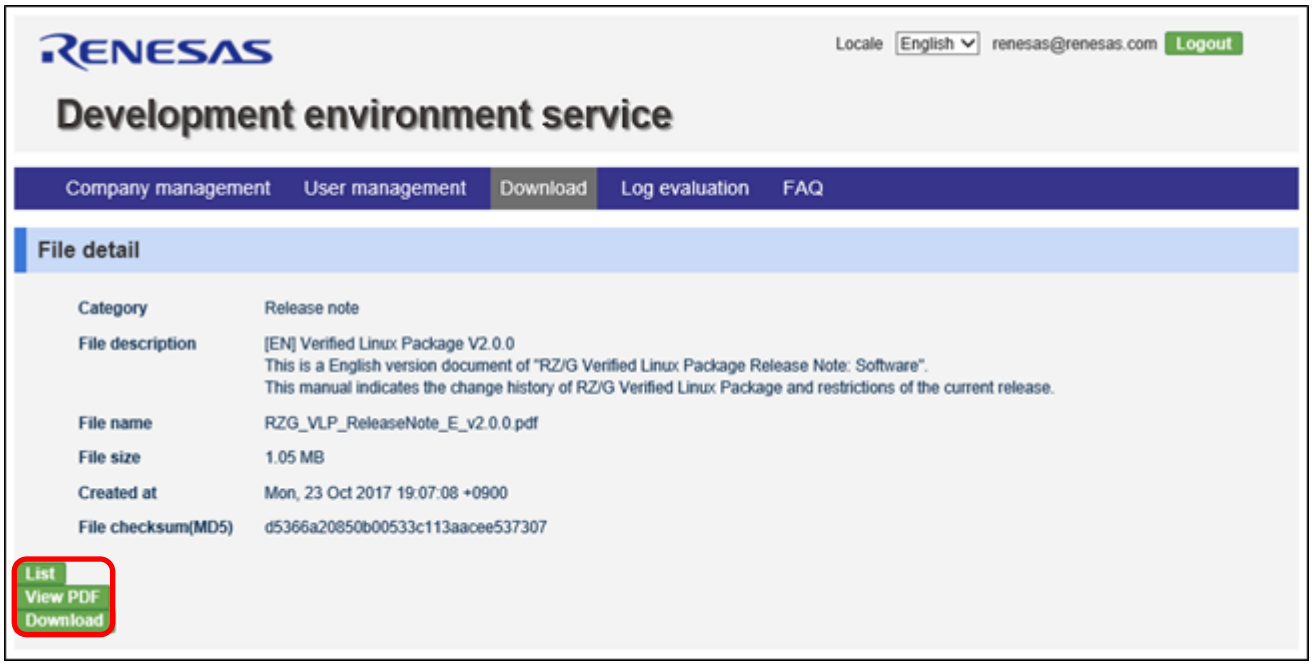
2. Clicking on [Download] for individual files in the [File list] window leads to downloading of the given files.



3. Clicking on a [Detail] placed on the right side of each file description in the [File list] window displays the [File detail] window of the given file.



4. Clicking on the [Download] button in the [File detail] window starts downloading of the file being displayed.



5. To return to the [File list] window, click on the [List] button.

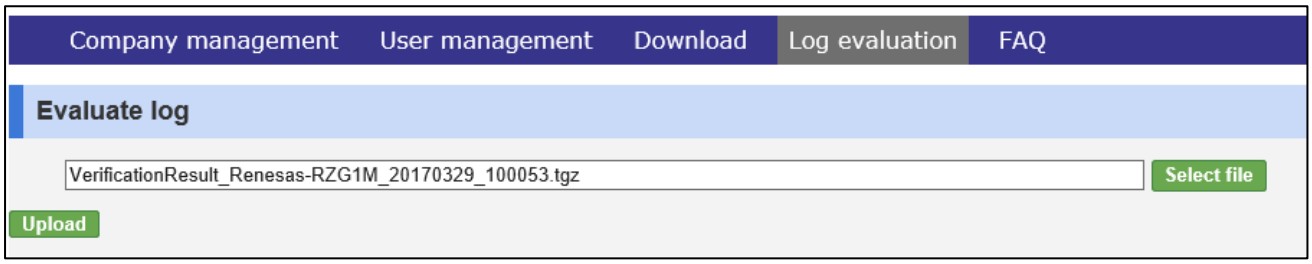
5.2 Handling the Analysis Tool

The following describes how to handle the analysis tool.

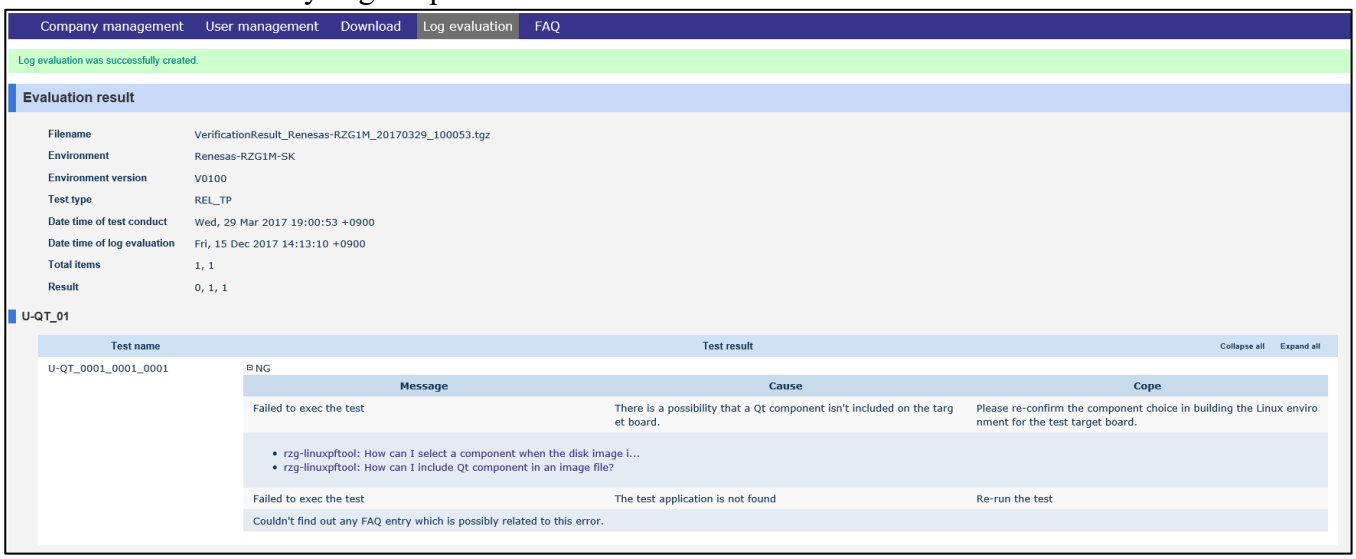
Handling the analysis tool:

1. Login to the web server by following the procedure given in section 5.1.2, and select [Log evaluation].

2. The [Evaluate log] screen is displayed. Specify the file exported for the analysis tool in section 4.4.4, and select [Upload].



3. The results of the file specified above are displayed. The information about any NG items is displayed. Specifically, this consists of a statement of the event that directly caused the item to be NG under [Message], the possible reason under [Cause], and measures to take under [Cope]. Related FAQ information is also displayed, with the intention of supporting the customer in analyzing the problems.



6. Handling the Traffic Monitor

The Verification tool can be used only when you purchased the Verification and Analysis tools.

6.1 Starting up the Traffic Monitor

Select [Show View] -> [Other] from the [Window] menu on the e^2 studio, and the screen shown in Figure 6-1 is displayed.
On the screen, select [RZ/G Linux Platform tools] -> [RZ/G Traffic monitor], and the traffic monitor window is displayed as shown in Figure 6-2.
Note that all mentions of menu items in this section refer to those in the window shown in Figure 6-2.

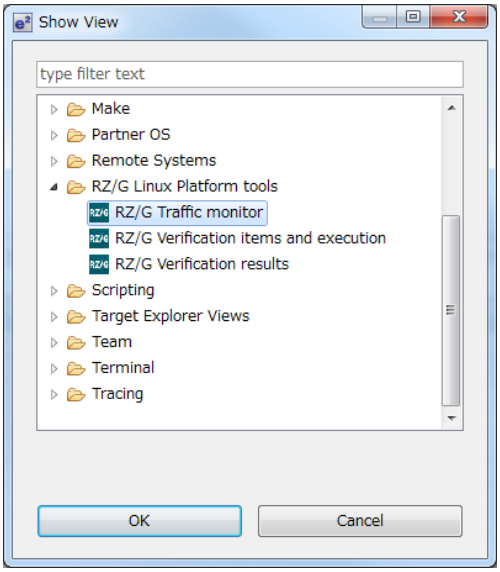


Figure 6-1 Screen for Selecting the Window to Display

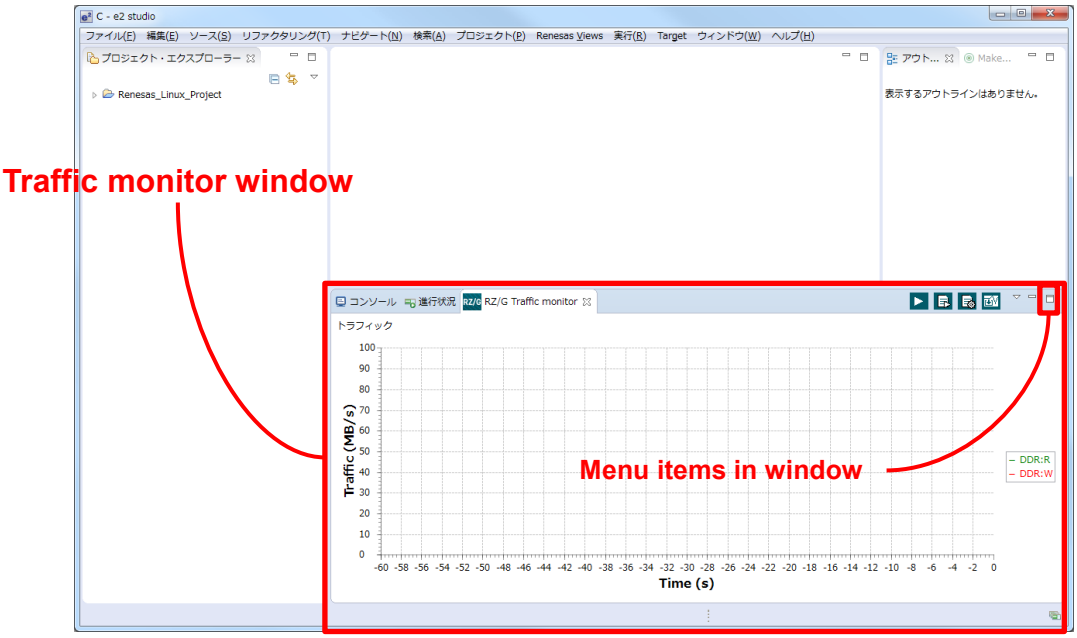


Figure 6-2 Traffic Monitor Window

6.2 Tool Settings

To use this tool, you will need to make the following settings to suit the environment in which you are using the tool.

1. Registering the Linux command for traffic monitoring
2. Settings for connection with the target board
3. Basic settings

Details of the items to be set are described on the following pages.

6.2.1 Registering the Linux command for traffic monitoring

Before using the traffic monitor, you will need to store an application program for measurement (the Linux command for traffic monitoring) in the target board.

To register the Linux command for traffic monitoring, proceed with either of the following steps.

- Select the [Register traffic monitor command to Embedded Linux project.] menu item from the [Setting] menu.
- Click on the [Register traffic monitor command to Embedded Linux project.] button on the toolbar.

Select the project for registering the Linux command for traffic monitoring and click on the [OK] button in [Select Embedded Linux Project] window shown in Figure 6-3. This registers the Linux command for traffic monitoring as an embedded Linux project as shown in Figure 6-4.

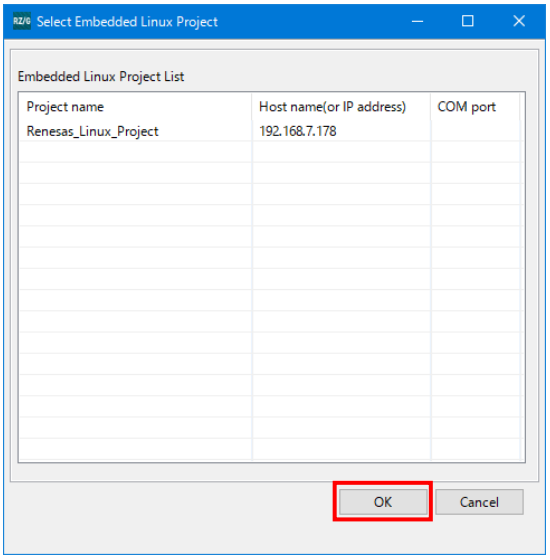


Figure 6-3 [Select Embedded Linux Project] Window

Figure 6-4 Result of Registering the Linux Command for Traffic Monitoring as an Embedded Linux Project

After the above processing, following the procedure below for the registered embedded Linux project allows incorporation of the Linux command for traffic monitoring.

1. Execute the Linux build to create a Linux boot image which incorporates the Linux command for traffic monitoring (see sections 3.8, 3.9, and 3.10).

2. Use the function for writing user programs to write the Linux command for traffic monitoring to the target board (see section 3.10).

6.2.2 Settings for Connection with the Target Board

For details on the settings for connection with the target board, see sections 4.2.1 and 4.2.2.

6.2.3 Basic Settings

Select [Preferences] from the [Window] menu of the e^2 studio. Select [RZ/G Linux Platform tools] -> [Traffic monitor] and make settings as described in Table 6-1. An example of the settings is shown in Figure 6-5.

Table 6-1 Basic Settings

Tab Name	Item to Be Set	Setting
Basic setting	Graph X Axis Max (s)	Specify the maximum amount of time that can be displayed along the x-axis of the graph. The default value is 60 seconds and the maximum specifiable value is 600 seconds.
	Execute by Ssh.	This selects whether connection to the target board is through a console or network (SSH connection via Ethernet).

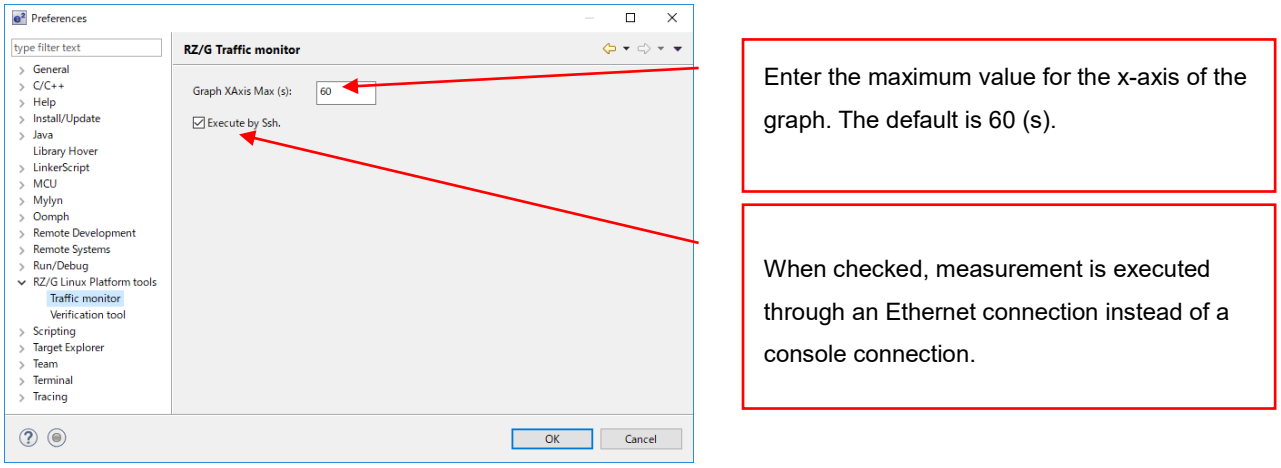


Figure 6-5 Basic Settings

6.3 Executing Measurement

6.3.1 Starting Measurement

Take either of the following steps to start measurement.

- Select the [Start] menu item from the [Measure] menu.
- Click on the [Start] button on the toolbar.

Once measurement starts, a graph is displayed to show the measured results in the [Traffic] section, as shown in Figure 6-6. A log of the results also appears in the [Console] section.

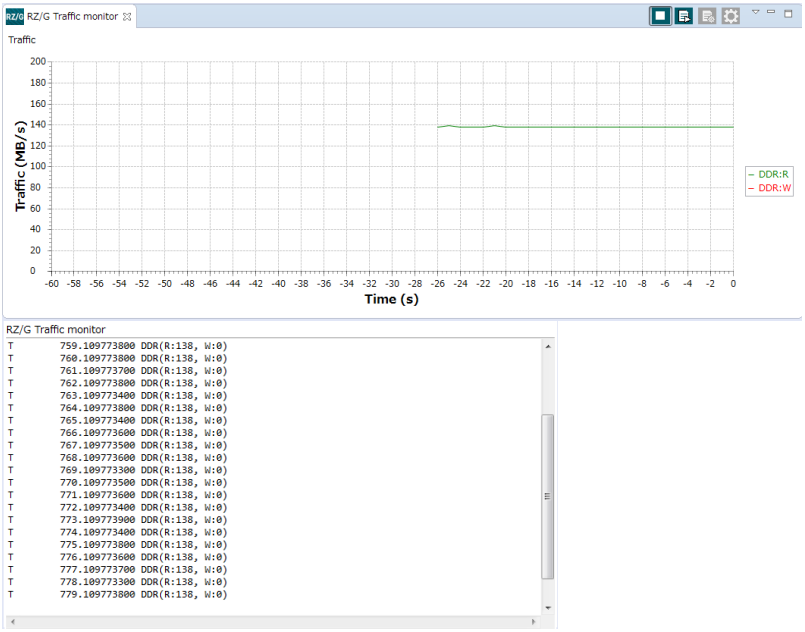


Figure 6-6 Display during Measurement

6.3.2 Stopping Measurement

To stop measurement while it is in progress, take one from among the following steps.

- Select the [Terminate] menu item from the [Measure] menu.
- Click on the [Terminate] button on the toolbar.
- Press any key on the keyboard.

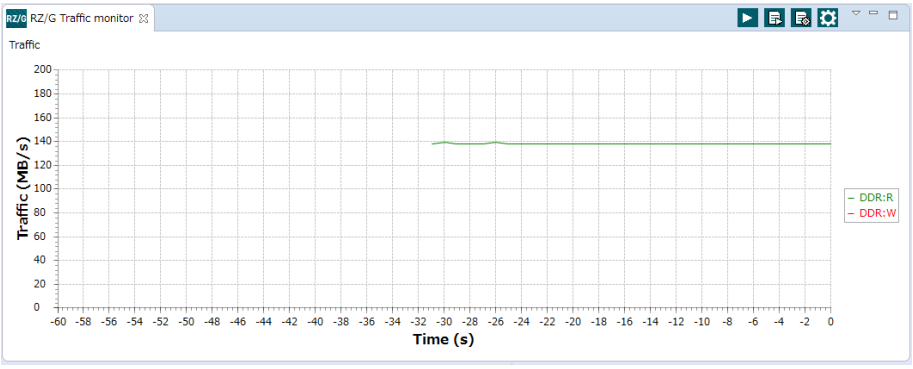


Figure 6-7 Display after Measurement Is Stopped

6.4 Saving a Log of Measured Results in a File

6.4.1 Starting the Log

Take either of the following steps to save the results of measurement in a file.

- Select the [Start] menu item from the [Log] menu.
- Click on the [Start Logging] button on the toolbar.

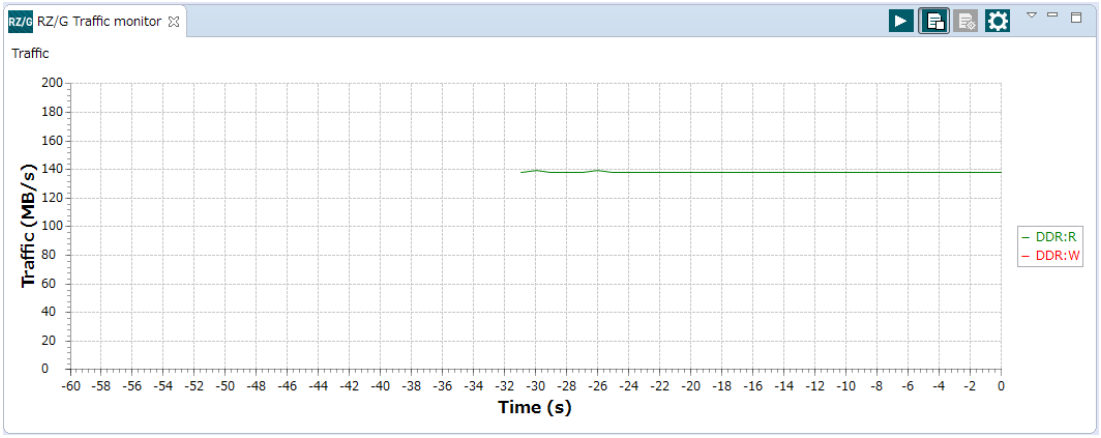


Figure 6-8 Display when Logging is Started

6.4.2 Stopping the Log

Take either of the following steps to stop saving the results of measurement in the file.

- Select the [Terminate] menu item from the [Log] menu.
- Click on the [Stop Logging] button on the toolbar.

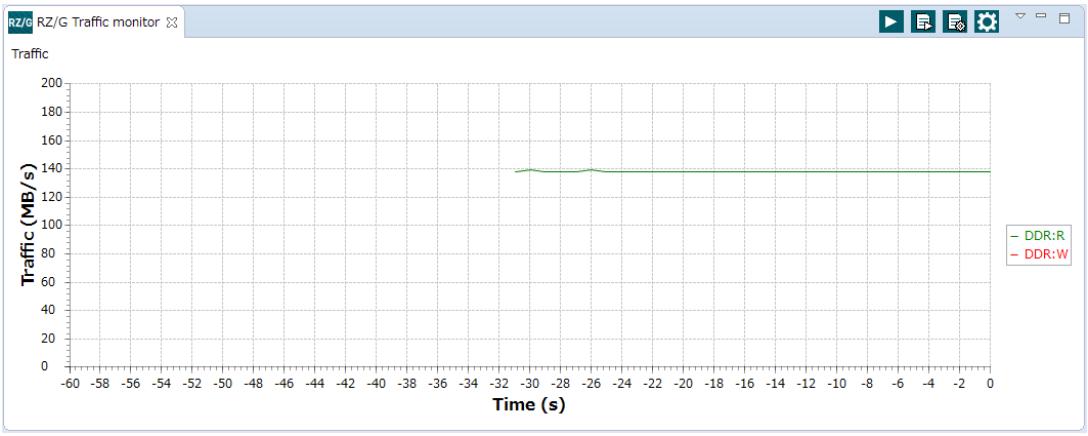


Figure 6-9 Display when Logging Is Stopped

6.4.3 Setting the Log File

Take either of the following steps to change the file for saving the results of measurement.

- Select the [Log file setting] menu item from the [Log] menu.
- Click on the [Log file setting] button on the toolbar.

Taking either of the steps produces the dialog box (Figure 6-10) for specifying the file for saving the log. Specify the filename in the dialog box.

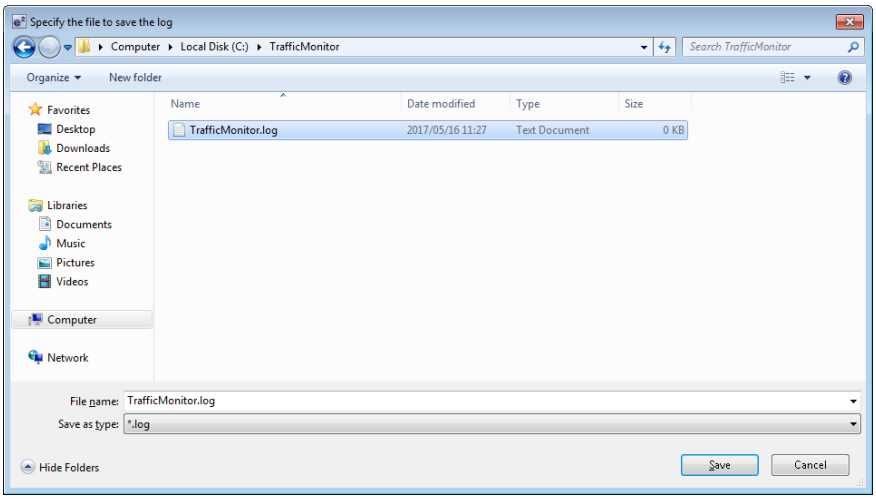


Figure 6-10 Dialog Box for Specifying the File for Saving the Log

6.4.4 Measured Data Saved in the Log File

As shown in Figure 6-11, the measured data are saved in text format in the log file.

The saved data consist of the log file header, information on the environment of the target board, marker for the point where measured data start, measured data, and marker for the point where the storage of measured data stops. The specific values measured are recorded in the measured data part.

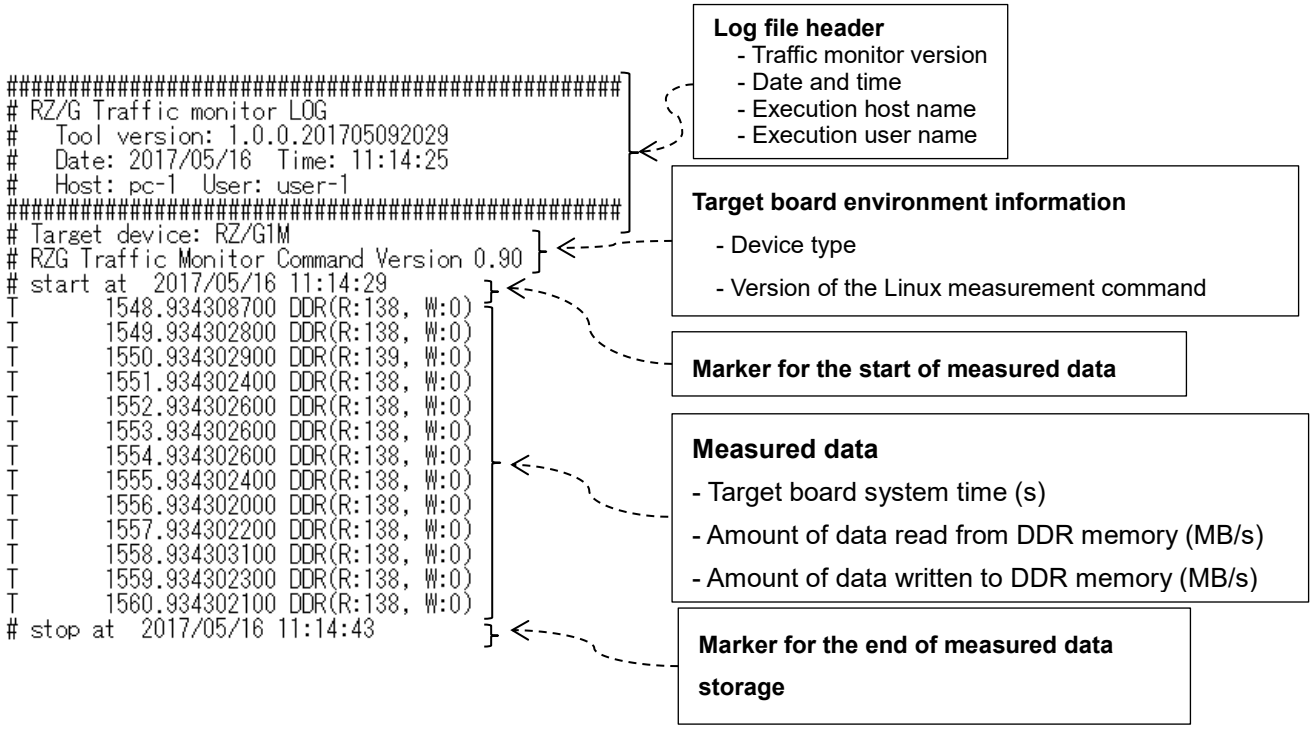


Figure 6-11 Measured Data Saved in a Log File

Revision History	RZ/G Linux Platform Development Environment User's Manual for the Linux Customization Tool, the Verification and Analysis Tools

Rev.	Date	Description	
		Page	Summary
2.00	Dec 2017	all	First revision issued. * Translated from Japanese document Rev2.00.
2.02	Aug 2017	-	Changed the document title.
		3	Section 2.1.1 changed the description for downloading e^2 studio 6.1.0.
		12	Section 2.1.5.1 changed the URL for downloading Nebula plugin.
		17 167 168 201 162-189	By adding the function for the test program parameter, added and changed as follows. - Step 4) of section 2.1.6 added the description for the folder storing parameters. - Table 4-2 added the description for the folder storing parameters. - Added the description to figure 4-7. - Added the description to section 4.10. - Changed following figure. Figure 4-2, 4-12, 4-13, 4-14, 4-16, 4-28, 4-29, 4-33, 4-36, 4-40
		65-72	Section 3.7.3.4 modified the description for samples and added Makefile samples.
		117	Step 2 of section 3.12.1 changed by adding the description for "overview" in "Modify kernel configs" window.
		118-122	Adding section "3.12.2. Adding and deleting of the Kernel Configuration function Kernel".
		126-128	Added description for "full custom mode" used to edit kernel code, to section 3.12.4.
		152	Section 3.15.1 changed the title and the description according to the tool menu.
		152	Step 2 of section 3.15.1 changed the figure and the description according to tool specification.
		155-159	Added section 3.16 titled as "Creating and Changing an SDK".
		160-161	Added section 3.17 titled as "Editing the "local.conf"".
		168	Figure 4-7 changed the description for stored folder of add-on test programs.
		216	Figure 6-5 updated the basic settings screen for Traffic monitor as latest.
		211	Step 3 of section 5.1.3.5 changed the description as clicking "Detail".
2.10	Sep 2018	all	Changed the tool name and its descriptions.
		3	Added e^2 studio 7.0.0 as a support environment, to section 1.2.
		19-20	For section 2.2.1, 2.3, changed the operation opening the installation detail screen.
		134-137	Added section "3.12.6 Debugging the Kernel"
		146	Added section "3.13.3 Debugging the U-boot"
2.11	Oct 2018	208-215	Removed the descriptions related with "Adding users" in the section 5.1.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

© 2018 Renesas Electronics Corporation. All rights reserved.

Colophon 5.2

RZ/G Linux Platform Development Environment User's Manual
for the Linux Customization Tool, the Verification and Analysis Tools

Publication Date: Rev.2.11 Oct 22, 2018

Published by: Renesas Electronics Corporation

RZ/G Series

Renesas Electronics Corporation

R01US0342EJ0211