

CubeSuite+ V1.03.00

Integrated Development Environment

User's Manual: RL78 Design

Target Device

RL78 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>).

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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of 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. 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 should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended 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; and industrial robots etc.
"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

Renesas Electronics products are neither intended nor 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 damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, please evaluate 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. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may 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 should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be 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, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

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

How to Use This Manual

This manual describes the role of the CubeSuite+ integrated development environment for developing applications and systems for RL78 family, and provides an outline of its features.

CubeSuite+ is an integrated development environment (IDE) for RL78 family, integrating the necessary tools for the development phase of software (e.g. design, implementation, and debugging) into a single platform.

By providing an integrated environment, it is possible to perform all development using just this product, without the need to use many different tools separately.

Readers This manual is intended for users who wish to understand the functions of the CubeSuite+ and design software and hardware application systems.

Purpose This manual is intended to give users an understanding of the functions of the CubeSuite+ to use for reference in developing the hardware or software of systems using these devices.

Organization This manual can be broadly divided into the following units.

CHAPTER 1 GENERAL
CHAPTER 2 FUNCTIONS (Pin Configurator)
CHAPTER 3 FUNCTIONS (Code Generator)
APPENDIX A WINDOW REFERENCE
APPENDIX B OUTPUT FILES
APPENDIX C API FUNCTIONS
APPENDIX D INDEX

How to Read This Manual It is assumed that the readers of this manual have general knowledge of electricity, logic circuits, and microcontrollers.

Conventions

Data significance:	<u>Higher</u> digits on the left and lower digits on the right
Active low representation:	XXX (overscore over pin or signal name)
Note:	Footnote for item marked with Note in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numeric representation:	Decimal ... XXXX Hexadecimal ... 0xXXXX

Related Documents

The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Document Name	Document No.	
CubeSuite+ Integrated Development Environment User's Manual	Start	R2OUT2133E
	V850 Design	R2OUT2134E
	R8C Design	R2OUT2135E
	RL78 Design	This manual
	78K0R Design	R2OUT2137E
	78K0 Design	R2OUT2138E
	RX Coding	R2OUT0767E
	V850 Coding	R2OUT0553E
	Coding for CX Compiler	R2OUT2139E
	R8C Coding	R2OUT0576E
	RL78, 78K0R Coding	R2OUT2140E
	78K0 Coding	R2OUT2141E
	RX Build	R2OUT0768E
	V850 Build	R2OUT0557E
	Build for CX Compiler	R2OUT2142E
	R8C Build	R2OUT0575E
	RL78, 78K0R Build	R2OUT2143E
	78K0 Build	R2OUT0783E
	RX Debug	R2OUT2175E
	V850 Debug	R2OUT2144E
	R8C Debug	R2OUT0770E
	RL78 Debug	R2OUT2145E
	78K0R Debug	R2OUT0732E
78K0 Debug	R2OUT0731E	
Analysis	R2OUT2146E	
Message	R2OUT2147E	

Caution The related documents listed above are subject to change without notice. Be sure to use the latest edition of each document when designing.

TABLE OF CONTENTS

CHAPTER 1 GENERAL ... 7

- 1.1 Overview ... 7
- 1.2 Features ... 7

CHAPTER 2 FUNCTIONS (Pin Configurator) ... 8

- 2.1 Overview ... 8
- 2.2 Open Device Pin List Panel ... 10
 - 2.2.1 Select item ... 11
 - 2.2.2 Change display order ... 12
 - 2.2.3 Add column ... 13
 - 2.2.4 Delete column ... 13
- 2.3 Open Device Top View Panel ... 14
 - 2.3.1 Select shape of microcontroller ... 15
 - 2.3.2 Select color ... 16
 - 2.3.3 Select popup information ... 17
 - 2.3.4 Select additional information ... 18
- 2.4 Enter Information ... 19
- 2.5 Output Report Files ... 20
 - 2.5.1 Output device pin list ... 20
 - 2.5.2 Output device top view ... 21

CHAPTER 3 FUNCTIONS (Code Generator) ... 22

- 3.1 Overview ... 22
- 3.2 Open Code Generator Panel ... 23
- 3.3 Enter Information ... 24
 - 3.3.1 Input rule ... 24
 - 3.3.2 Icon indicating incorrect entry ... 25
 - 3.3.3 Icon indicating pin conflict ... 26
- 3.4 Confirm Source Code ... 27
- 3.5 Output Source Code ... 28
 - 3.5.1 Set whether or not to generate source code ... 29
 - 3.5.2 Change file name ... 30
 - 3.5.3 Change API function name ... 31
 - 3.5.4 Change output mode ... 32
 - 3.5.5 Change output destination folder ... 33
- 3.6 Output Report Files ... 34
 - 3.6.1 Change output format ... 36
 - 3.6.2 Change output destination ... 37

APPENDIX A WINDOW REFERENCE ... 38

A.1 Description ... 38

APPENDIX B OUTPUT FILES ... 84

B.1 Overview ... 84

B.2 Output File ... 84

APPENDIX C API FUNCTIONS ... 91

C.1 Overview ... 91

C.2 Output Function ... 91

C.3 Function Reference ... 101

C.3.1 Clock Generator ... 103

C.3.2 Port ... 110

C.3.3 Interrupt ... 113

C.3.4 Serial ... 127

C.3.5 A/D Converter ... 214

C.3.6 D/A Converter ... 229

C.3.7 Timer ... 236

C.3.8 Watchdog Timer ... 294

C.3.9 Real-time Clock ... 299

C.3.10 Interval Timer ... 320

C.3.11 Comparator ... 327

C.3.12 Clock Output/Buzzer Output ... 334

C.3.13 Data Transfer Controller ... 340

C.3.14 Event Link Controller ... 346

C.3.15 LCD Controller/Driver ... 350

C.3.16 Sound Generator ... 359

C.3.17 DMA Controller ... 365

C.3.18 Voltage Detector ... 374

C.3.19 Programmable Gain Amplifier ... 379

APPENDIX D INDEX ... 384

CHAPTER 1 GENERAL

CubeSuite+ is an integrated development environment used to carry out tasks such as design, coding, build and debug for developing application systems.

This chapter gives an overview of the design tool (Pin Configurator/Code Generator).

1.1 Overview

The design tool, which is one of the components provided by CubeSuite+, enables you to output the pin assignment of the microcontroller (device pin list and device top view), and the source code (device driver programs, C source files and header files) necessary to control the peripheral functions provided by the microcontroller (clock generator, port functions, etc.) by configuring various information using the GUI.

1.2 Features

The design tool (Pin Configurator/Code Generator) has the following features.

- Code generating function

The Code Generator can output not only device driver programs in accordance with the information configured using the GUI, but also a build environment such as sample programs containing main functions and link directive files.

- Reporting function

You can output configured information using the Pin Configurator/Code Generator as files in various formats for use as design documents.

- Renaming function

The user can change default names assigned to the files output by the Code Generator and the API functions contained in the source code.

CHAPTER 2 FUNCTIONS (Pin Configurator)

This chapter describes the key functions provided by the design tool (Pin Configurator) along with operation procedures.

2.1 Overview

The Pin Configurator is used to output report files such as a device pin list and a device top view by entering pin assignment information of the microcontroller.

The following sections describe the operation procedures for Pin Configurator.

(1) Start CubeSuite+

Launch CubeSuite+ from the [Start] menu of Windows.

Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Start CubeSuite+".

(2) Create/Open project

Create a new project (that defines a kind of project, microcontroller to be used, build tools to be used, etc.) or load an existing project.

Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Create/Open project".

(3) Open Device Pin List Panel

Open the [Device Pin List panel](#), where you enter information on the pins of the microcontroller.

(a) Select item

Allows you to select items displayed in the device pin list.

(b) Change display order

Allows you to change the order in which items are displayed in the device pin list.

(c) Add column

Allows you to add columns to the device pin list.

(d) Delete column

Allows you to delete columns from the device pin list.

(4) Open Device Top View Panel

Open the [Device Top View panel](#), where you can confirm the information entered for the pins.

(a) Select shape of microcontroller

Allows you to select the shape of the microcontroller displayed in the [Device Top View panel](#).

(b) Select color

Allows you to select colors used to distinguish the type of pins (power pins, special pins, used pins, etc.) whose information is displayed in the [Device Top View panel](#).

(c) Select popup information

Allows you to select the type of information that pops up when you move the mouse cursor over each pin in the [Device Top View panel](#).

(d) Select additional information

Select the type of information to display in Pin area of the [Device Top View panel](#).

(5) Enter Information

Enter information on the pins of the microcontroller in the [Device Pin List panel](#).

(6) Output Report Files

Output report files (files containing configured information using Pin Configurator: device pin list and device top view) to the specified folder.

(a) Output device pin list

Output a device pin list.

(b) Output device top view

Output a device top view.

(7) Save project

Save a project.

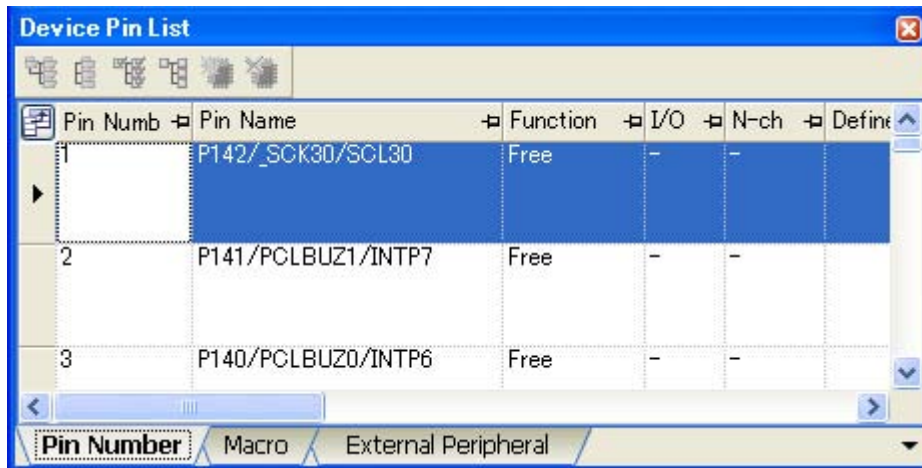
Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Save project".

2.2 Open Device Pin List Panel

Open the [Device Pin List panel](#), where you enter information on the pins of the microcontroller.


To open the [Device Pin List panel](#), double-click [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List] in the [Project Tree panel](#).

Figure 2-1. Open Device Pin List Panel



- Remarks 1.** If an unsupported microcontroller is defined in the project for Pin Configurator, then "[Pin Configurator (Design Tool)] node" will hide under [*Project name* (Project)] in the [Project Tree panel](#).
- 2.** The [Device Pin List panel](#) consists of three tabs. Selecting one of the tabs changes the order in which "information on each pin of the microcontroller" is displayed.
- [\[Pin Number\] tab](#)
Information on each pin of the microcontroller is displayed in the order of pin number.
 - [\[Macro\] tab](#)
Information on each pin of the microcontroller is displayed in the order it was grouped into peripheral functions.
 - [\[External Peripheral\] tab](#)
Information about the pins connected to external peripherals is displayed in order grouped at the external-peripheral component level.

2.2.1 Select item

The Pin Configurator is used to select items to be displayed in the device pin list using the  button in the upper left corner of the device pin list.


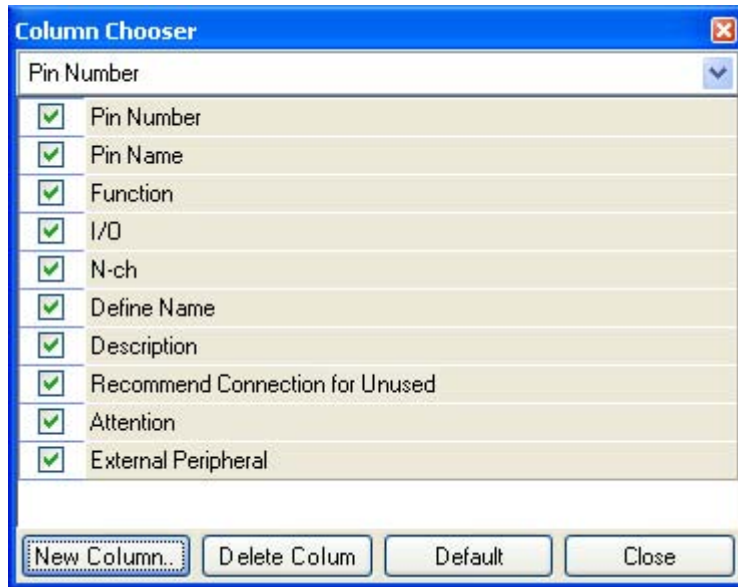
To select the item to be displayed, use the [Column Chooser dialog box](#) that opens by pressing the  button in the upper left corner of the device pin list.

Figure 2-2. Select Item



Remark To select the item to be displayed, check the check box that corresponds to the item.

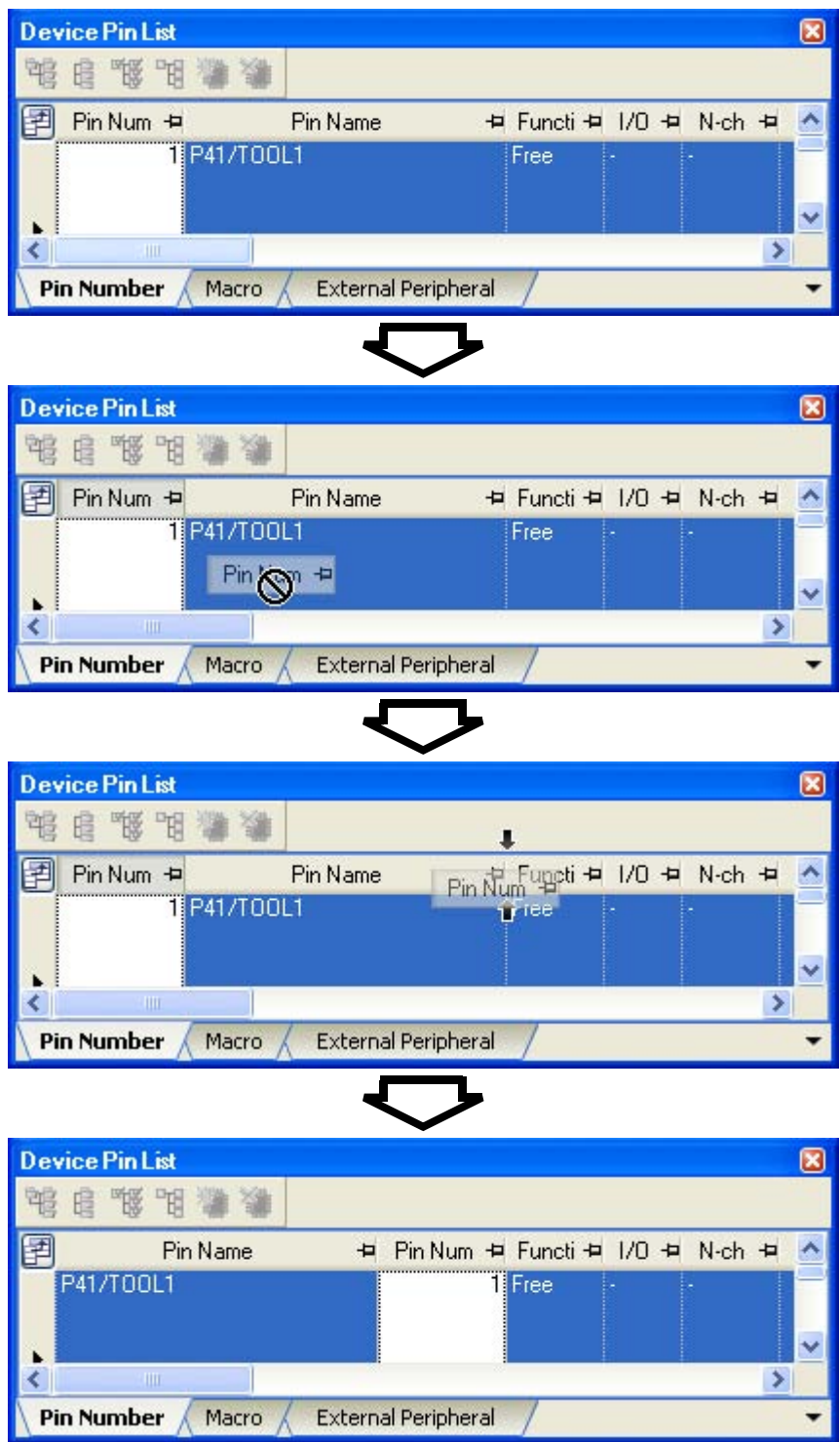
Table 2-1. Select Item


Checked	Displays the selected item in the device pin list.
Not checked	Hides the selected item in the device pin list.

2.2.2 Change display order

In Pin Configurator, you can change the display order of columns in the device pin list (move columns) by dragging and dropping columns.

Figure 2-3. Change Display Order



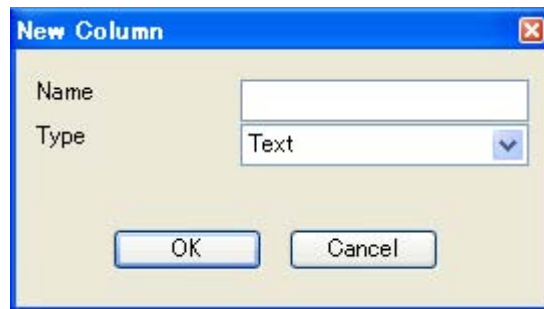
Remark To change the display order, click the  button in the upper left of the device pin list. The [Column Chooser dialog box](#) opens. Drag an item displayed in the dialog's select Items to display area, and drop it to the desired destination in the device pin list. This will change the display order.

2.2.3 Add column

The Pin Configurator is used to add the "user's own column" to the device pin list using the [New Column...] button in the [Column Chooser dialog box](#) that opens by pressing the  button in the upper left corner of the device pin list.


To add a column, use the [New Column dialog box](#) that opens by pressing the [New Column...] button in the [Column Chooser dialog box](#).

Figure 2-4. Add Column



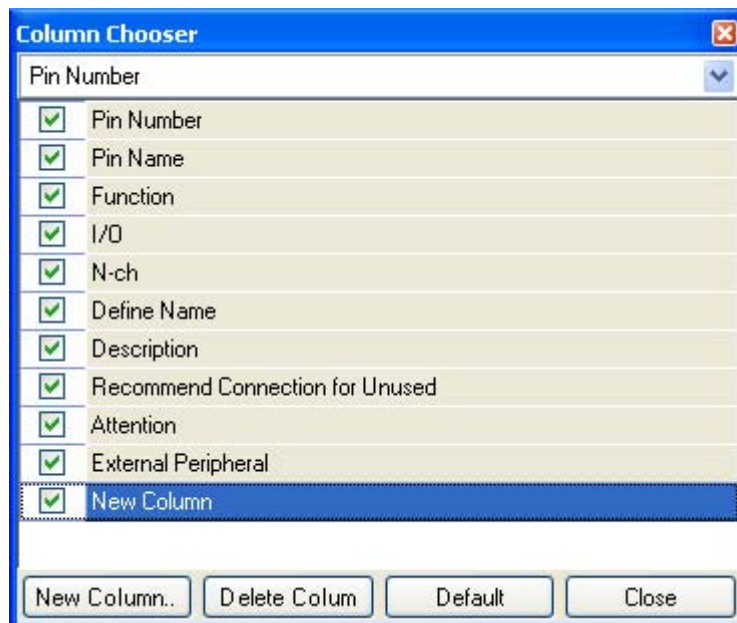
Remark On the device pin list, adding columns to the first level of [\[Macro\] tab](#), [\[External Peripheral\] tab](#) is restricted.

2.2.4 Delete column

The Pin Configurator is used to delete the "user's own column" from the device pin list using the [Delete Column] button in the [Column Chooser dialog box](#) that opens by pressing the  button in the upper left corner of the device pin list.

To delete a column, select the column you want to delete in the displayed item selection area of the [Column Chooser dialog box](#), and press the [Delete Column] button.

Figure 2-5. Delete Column



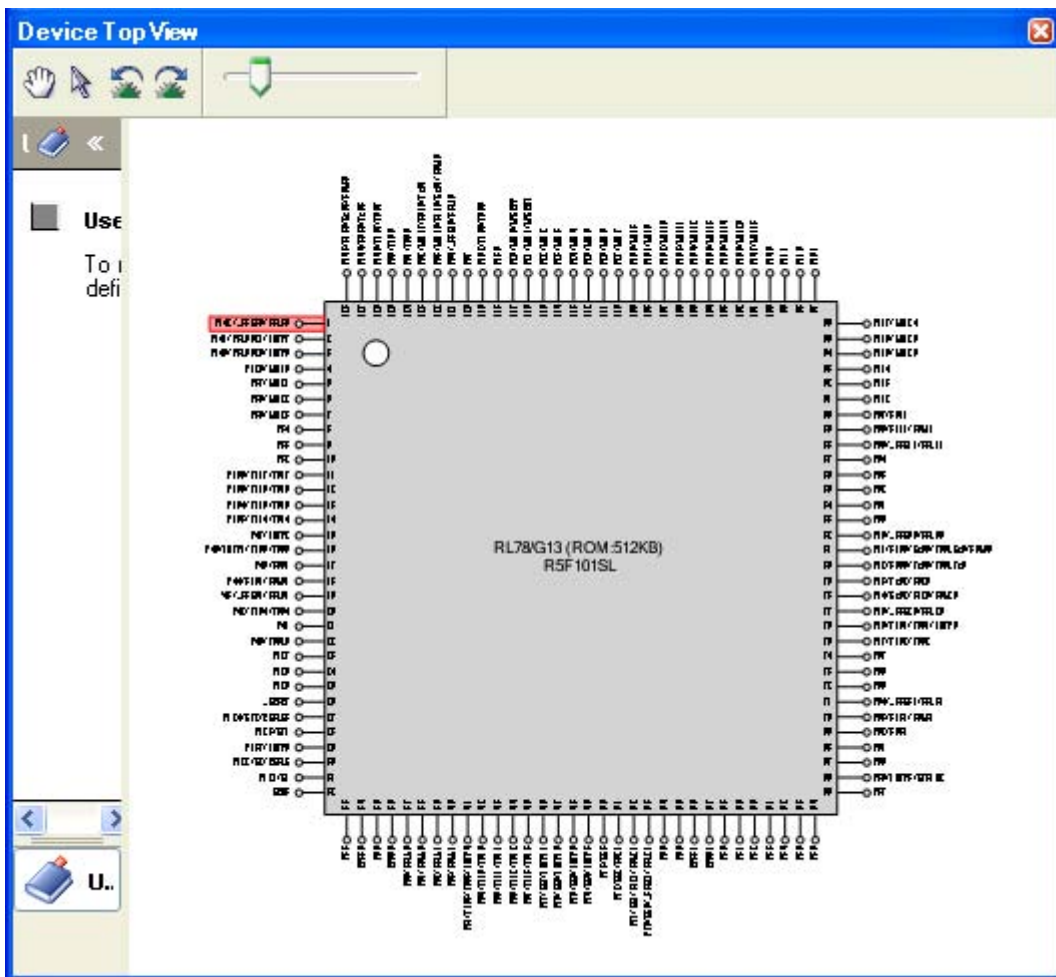
Remark You can only delete the column which you added using the [New Column dialog box](#).

2.3 Open Device Top View Panel

Open the [Device Top View panel](#), where you can confirm the information entered for the pins of the microcontroller.

To open the [Device Top View panel](#), double-click [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Top View] in the [Project Tree panel](#).

Figure 2-6. Open Device Top View Panel



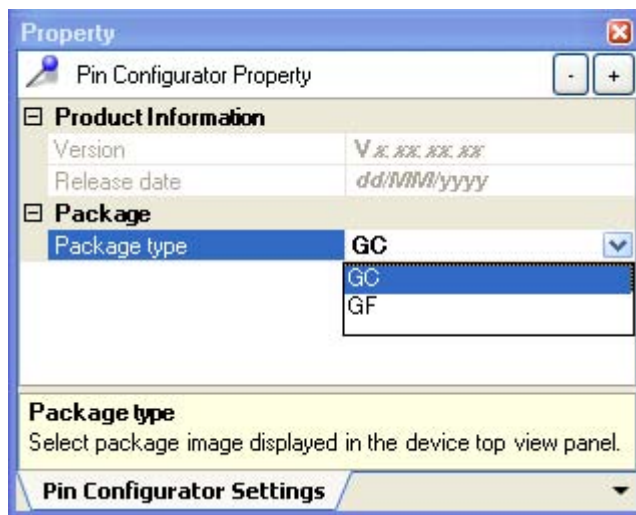
Remark In the [Property panel](#), on the [[Pin Configurator Settings](#)] tab, if "BGA" is selected for the Package type, then [Device Top View panel](#) cannot be opened.

2.3.1 Select shape of microcontroller

Select the shape of the microcontroller displayed in the [Device Top View panel](#) which is opened as described in "2.3 [Open Device Top View Panel](#)".

To select the shape of the microcontroller, click [[Pin Configurator Settings](#)] tab >> [Package type] in the [Property panel](#) and select the desired shape.

Figure 2-7. Select Shape of Microcontroller



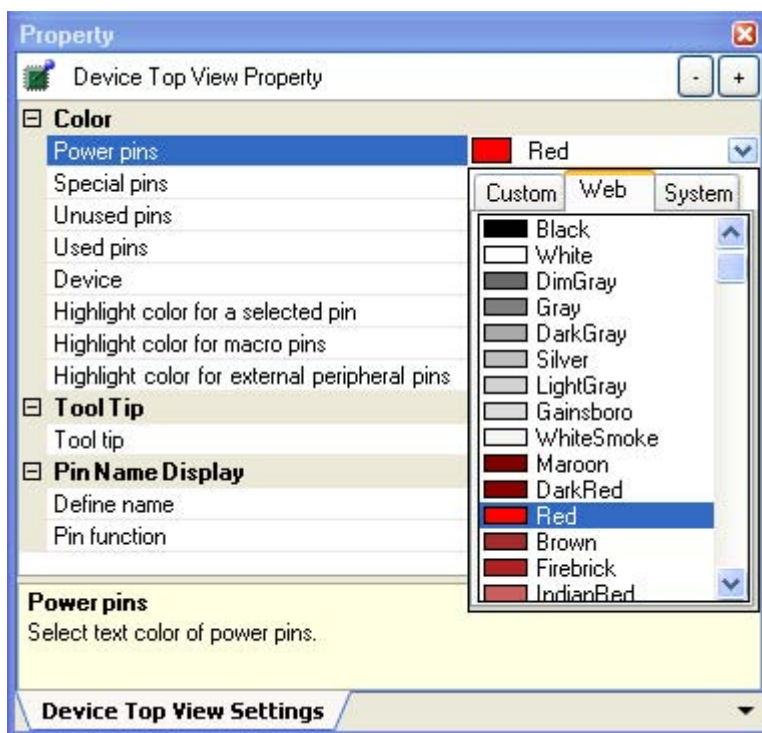
Remark Selection of the shape of the microcontroller is made using the order name (such as GC and GF).

2.3.2 Select color

Select the colors used to distinguish the type of pins (power pins, special pins, unused pins, etc.) whose information is displayed in the [Device Top View panel](#) which is opened as described in "2.3 Open Device Top View Panel".

To select the color to be displayed, select the desired color in the color palette that opens by clicking [[Device Top View Settings](#)] tab >> [Color] in the [Property panel](#).

Figure 2-8. Select Color



Remark Select the colors to be displayed for the following eight types of items.

Table 2-2. Select Color

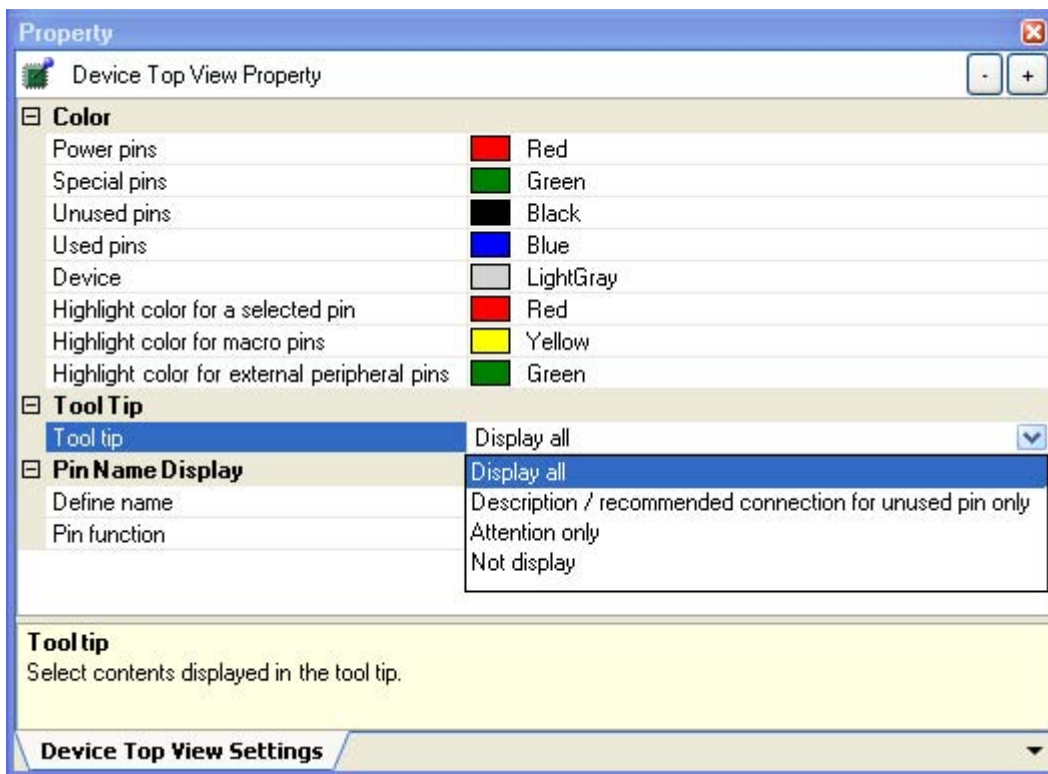
Item	Outline
Power pins	Selects the display color for power pins (pins whose use is limited to power).
Special pins	Selects the display color for special pins (pins with specified uses).
Unused pins	Selects the display color for unused pins (dual-use pins with no use set in the Device Pin List panel).
Used pins	Selects the display color for used pins (dual-use pins with a use set in the Device Pin List panel).
Device	Selects the display color of the microcontroller.
Highlight color for a selected pin	Selects the background color of a pin selected in the Device Pin List panel , on the [Pin Number] tab.
Highlight color for macro pins	Selects the background color of pins selected in the Device Pin List panel , on the [Macro] tab.
Highlight color for external peripheral pins	Selects the background color of pins selected in the Device Pin List panel , on the [External Peripheral] tab.

2.3.3 Select popup information

Select the type of information that popups when you move the mouse cursor over each pin in the [Device Top View panel](#) which is opened as described in "2.3 Open Device Top View Panel".

To select the popup information, click [[Device Top View Settings](#)] tab >> [Tool tip] in the [Property panel](#) and select the desired type of information.

Figure 2-9. Select Popup Information



Remark Popup information is selected from the following four types.

Table 2-3. Select Popup Information

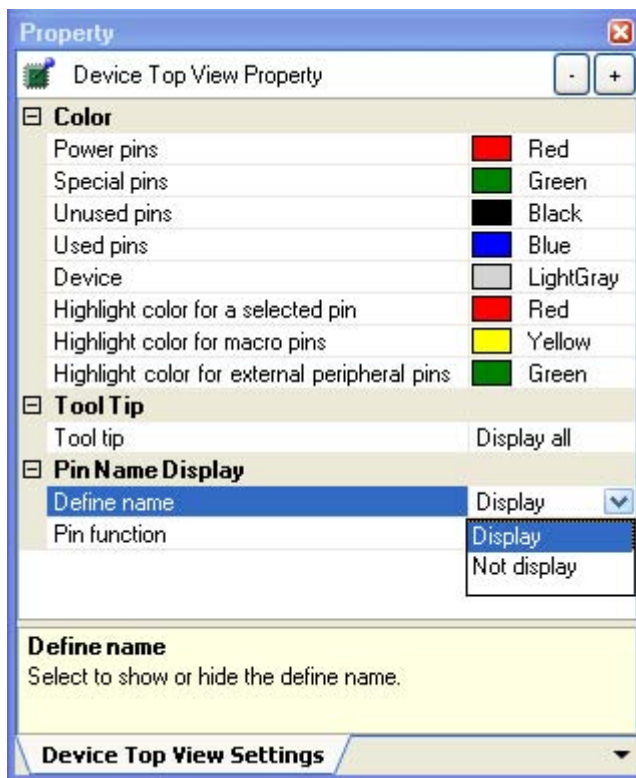
Popup Information	Outline
Display all	Displays the "Description", "Recommend Connection for Unused", and "Attention" strings for the device pin list.
Description / recommended connection for unused pin only	Displays the "Description", and "Recommend Connection for Unused" strings for the device pin list.
Attention only	Displays the "Attention" string for the device pin list.
Not display	Hides tooltips when the mouse cursor hovers over a pin.

2.3.4 Select additional information

Select the type of information to display in Pin area, in the [Device Top View panel](#) opened in "2.3 Open Device Top View Panel".

Note that additional information is selected from the [Property panel](#), on the [\[Device Top View Settings\] tab](#), by selecting the corresponding information under [\[Pin Name Display\]](#).

Figure 2-10. Select Additional Information



Remarks 1. Select one of the following two types for Define name (whether to display the "Define Name" string of the Device Pin List in appended format).

Display	Displays the "Define Name" string of the device pin list in appended format.
Not display	Hides the "Define Name" string of the device pin list.

2. Select one of the following two types for Pin function (whether to display it whether or not a function is selected for "Function" on the Device Pin List).

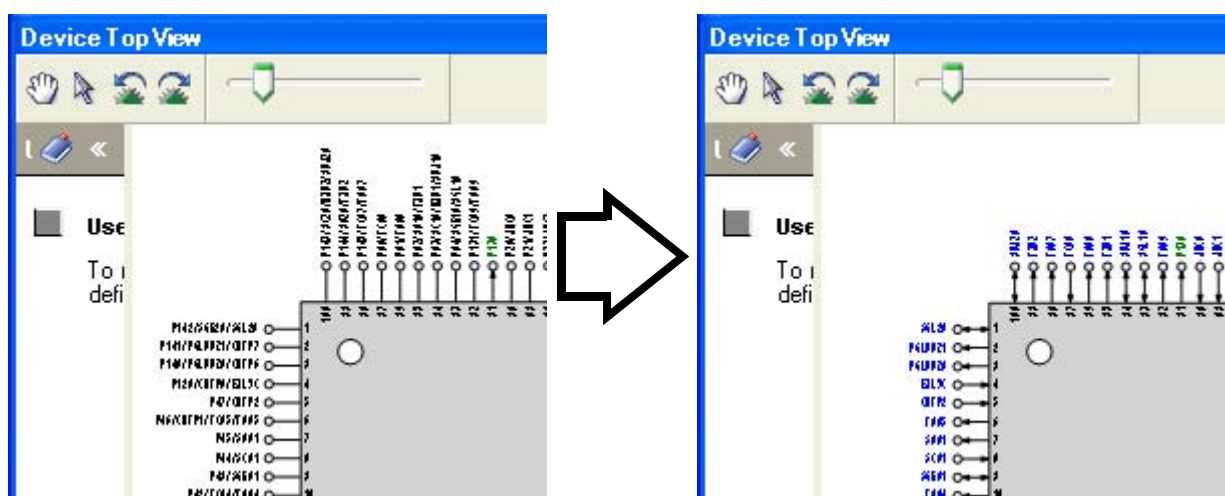
Display all	Displays functions selected via the device pin list's "Function" feature in parentheses.
Selected function only	Only display functions selected via the device pin list's "Function" feature in the device top view.

2.4 Enter Information

Enter information on the pins of the microcontroller in the [Device Pin List panel](#) which is opened as described in "2.2 [Open Device Pin List Panel](#)".

- Remarks 1.** You cannot add information in the "Pin Number" column, "Pin Name" column, "Description" column, "Recommend Connection for Unused" column and "Attention" column because they contain fixed information.
- 2.** If the "Free" in the "Function" column is changed to a specific pin name, color of the corresponding pin in the [Device Top View panel](#) changes from the "color representing the unused pins" to the "color representing the used pins" selected by clicking [[Device Top View Settings](#)] tab >> [Color] in the [Property panel](#).

Figure 2-11. Change in Displayed Color



2.5 Output Report Files

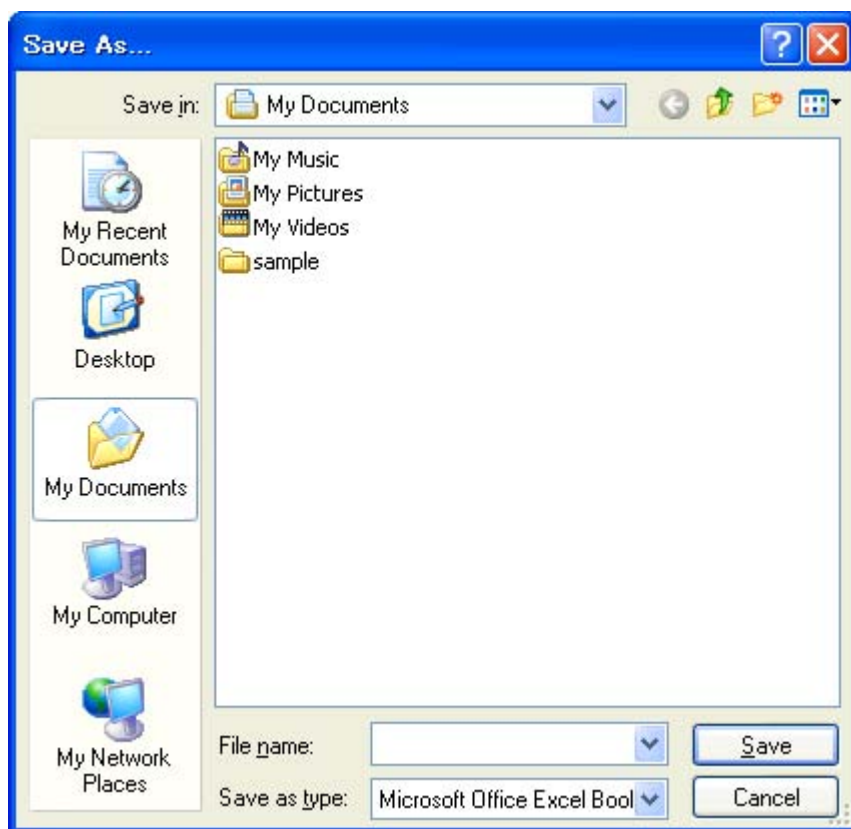
Output report files (files containing information configured using Pin Configurator: device pin list and device top view) to the specified folder.

2.5.1 Output device pin list

Select [File] menu >> [Save Pin List As...] to output a report file (a file containing information configured using Pin Configurator: device pin list).

The destination folder for the device pin list is specified in the [Save As dialog box](#) which opens by selecting [File] menu >> [Save Pin List As ...].

Figure 2-12. Output Device Pin List



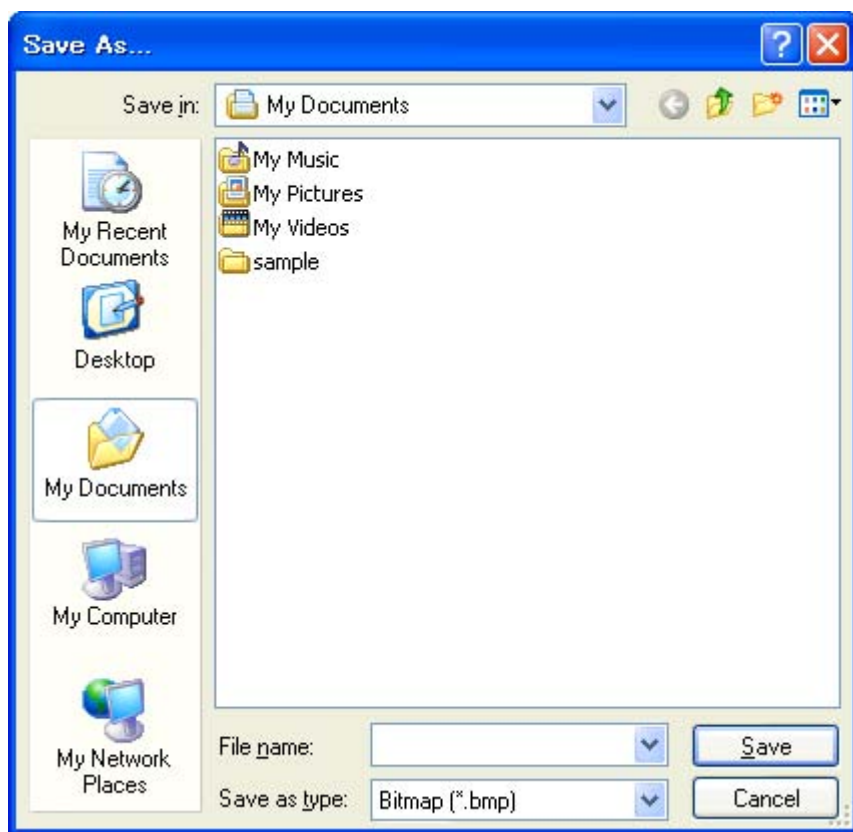
- Remarks 1.** If a device pin list has been already output, that list will be overwritten by selecting [File] menu >> [Save Pin List].
- 2.** The output format for the device pin list is limited to Microsoft Office Excel Book.

2.5.2 Output device top view

Select [File] menu >> [Save Top View As...] to output a report file (a file containing information configured using Pin Configurator: device top view).

The destination folder for the device top view is specified in the [Save As dialog box](#) which opens by selecting [File] menu >> [Save Top View As ...].

Figure 2-13. Output Device Top View



Remark If a device top view has been already output, that view will be overwritten by selecting [File] menu >> [Save Top View].

CHAPTER 3 FUNCTIONS (Code Generator)

This chapter describes the key functions provided by the design tool (Code Generator) along with operation procedures.

3.1 Overview

The Code Generator outputs source code (device driver programs) based on information selected/entered on CubeSuite+ panels that is needed to control peripheral functions provided by the microcontroller (clock generator, port functions, etc.).

The following sections describe the operation procedures for Code Generator.

(1) Start CubeSuite+

Launch CubeSuite+ from the [Start] menu of Windows.

Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Start CubeSuite+".

(2) Create/Open project

Create a new project (that defines a kind of project, microcontroller to be used, build tools to be used, etc.) or load an existing project.

Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Create/Open project".

(3) Open Code Generator Panel

Open the [Code Generator panel](#) used to configure the information necessary to control the peripheral functions (clock generator, port functions, etc.).

(4) Enter Information

Configure the information necessary to control the peripheral functions in the [Code Generator panel](#).

(5) Confirm Source Code

Confirm the source code (device driver program) that reflects the information configured in the [Code Generator panel](#).

(6) Output Source Code

Output the source code (device driver program) to the specified folder.

(7) Output Report Files

Output report files (a file containing information configured using Code Generator and a file containing information regarding the source code) to the specified folder.

(8) Save project

Save a project.

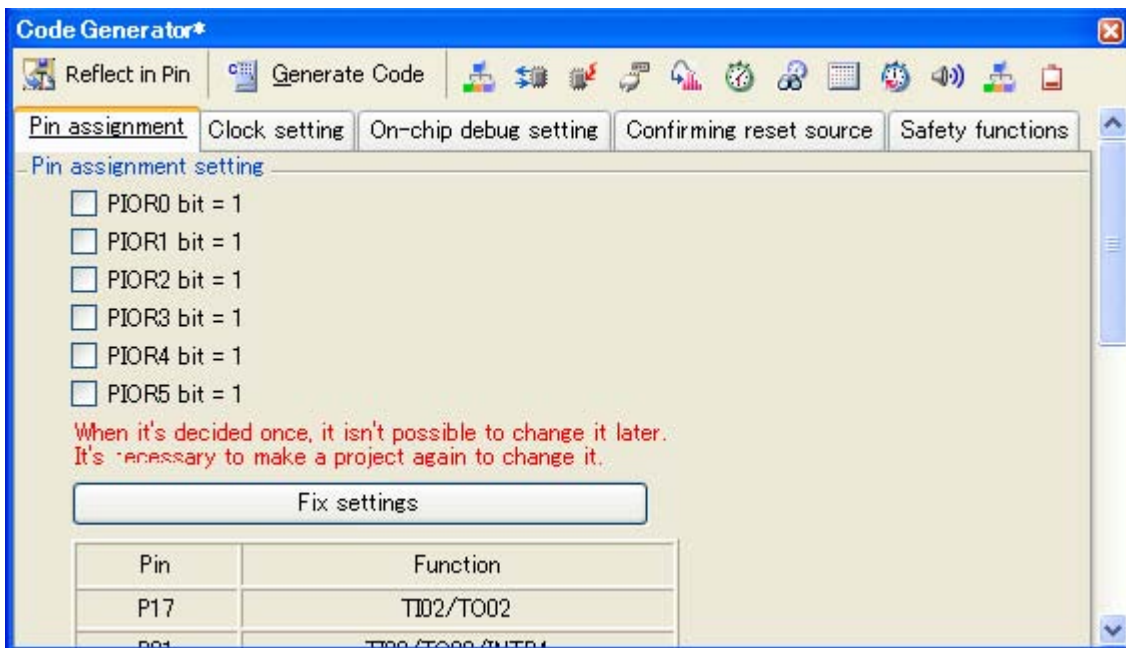
Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Save project".

3.2 Open Code Generator Panel

Open the [Code Generator panel](#) to configure the information necessary to control the peripheral functions (clock generator, port functions, etc.).

To open the [Code Generator panel](#), double-click [*Project name* (Project)] >> [Code Generator (Design Tool)] >> Peripheral function node "[Clock Generator], [Port], etc." in the [Project Tree panel](#).

Figure 3-1. Open Code Generator Panel



Remark If an unsupported microcontroller is defined in the project for Code Generator, then "[Code Generator (Design Tool)] node" will hide under [*Project name* (Project)] in the [Project Tree panel](#).

3.3 Enter Information

Configure the information necessary to control the peripheral functions in the information setting area of the [Code Generator panel](#) which is opened as described in "3.2 [Open Code Generator Panel](#)".

Remark When controlling multiple peripheral functions, repeat the procedures described in "3.2 [Open Code Generator Panel](#)" through "3.3 [Enter Information](#)".

3.3.1 Input rule

Following is the rules for input to the [Code Generator panel](#).

(1) Character set

Character sets that are allowed to input are as follows.

Table 3-1. List of Character Set

Character Set	Outline
ASCII	1-byte alphabet, number, symbol
Shift-JIS	2-byte alphabet, number, symbol, Hiragana, Katakana, Kanji and 1-byte Katakana
EUC-JP	2-byte alphabet, number, symbol, Hiragana, Katakana, Kanji and 1-byte Katakana
UTF-8	2-byte alphabet, number, symbol, Hiragana, Katakana, Kanji (include Chinese character) and 1-byte Katakana


(2) Number

Notations allowed when entering numbers are as follows.

Table 3-2. List of Notation

Notation	Outline
Decimal number	A numeric value that starts with a number between 1 and 9 and followed by numbers between 0 and 9, and the numeric value 0
Hex number	A numeric value that starts with 0x and followed by a combination of numbers from 0 to 9 and characters from A to F (characters are not case sensitive)

3.3.2 Icon indicating incorrect entry

When performing code generation, if you enter an invalid string in the [Code Generator panel](#), or a required input is missing, then a  icon displays next to the incorrect input, and the text is displayed in red to warn that there is a problem with the input.


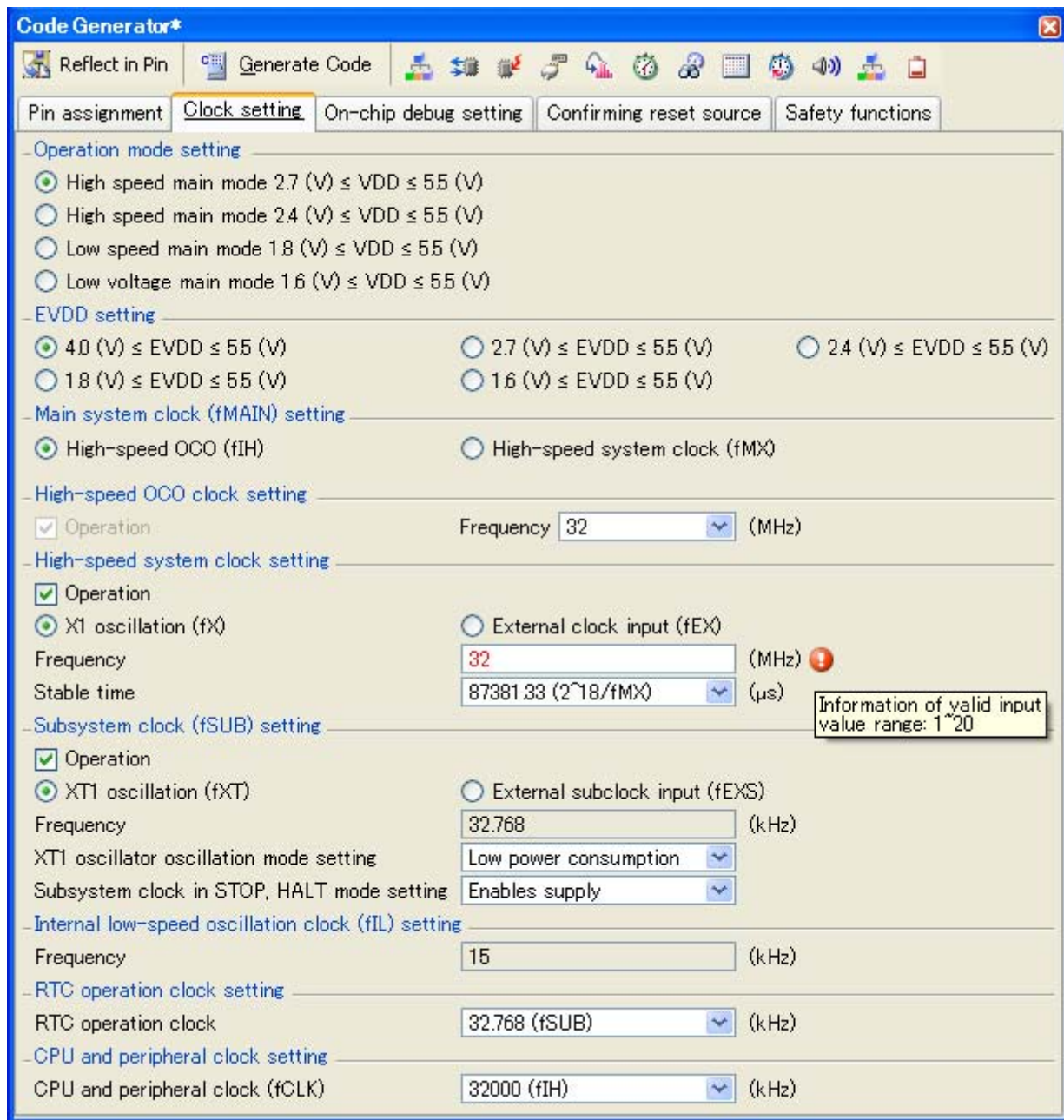

Remark If the mouse cursor is moved over the  icon, information regarding the string that should be entered (tips for correcting the entry) pops up.

Figure 3-2. Icon Indicating Incorrect Entry



3.3.3 Icon indicating pin conflict

If a conflict occurs between the pins while setting various peripheral functions in the [Code Generator panel](#), the  icon is displayed at the location where the conflict occurs to warn the user of a conflict between the pins.


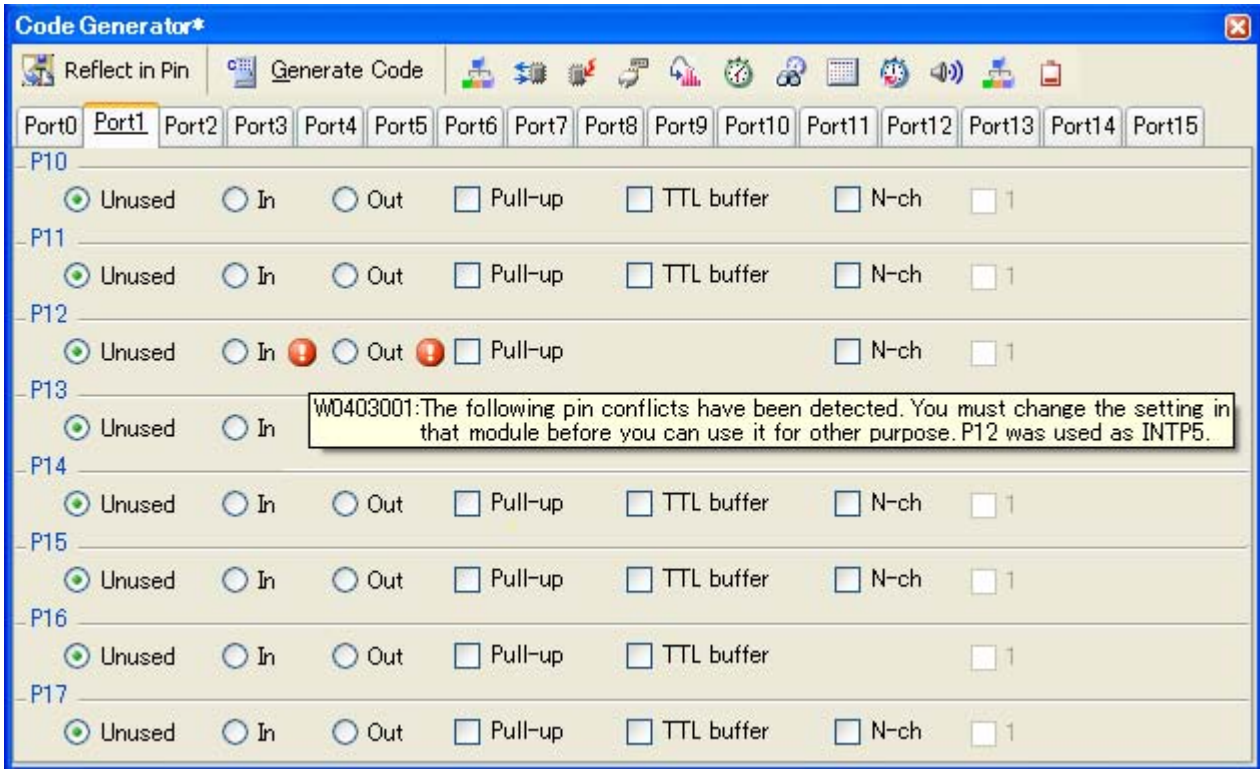
Remark If the mouse cursor is moved over the  icon, information regarding the conflict between the pins (tips for avoiding the conflict) pops up.

Figure 3-3. Icon Indicating Pin Conflict

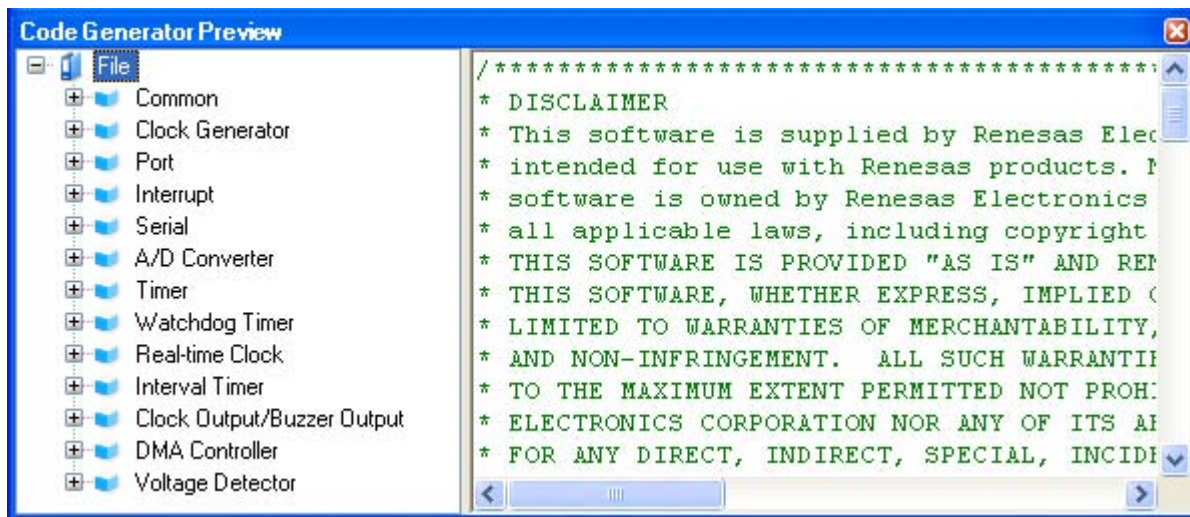


3.4 Confirm Source Code

Confirm the source code (device driver program) that reflects the information configured as described in "3.3 Enter Information".

To confirm the source code, use the [Code Generator Preview panel](#) that opens by selecting [View] menu >> [Code Generator Preview].

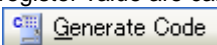
Figure 3-4. Confirm Source Code



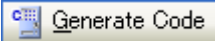
- Remarks 1. You can change the source code to be displayed by selecting the source file name or API function name in the [Code Generator Preview panel](#).
- 2. The following table displays the meaning of the color of the source code text displayed in the [Code Generator Preview panel](#).

Table 3-3. Color of Source Code

Color	Outline
Green	Comment
Blue	Reserved word for C compiler
Red	Numeric value
Black	Code section
Gray	File name

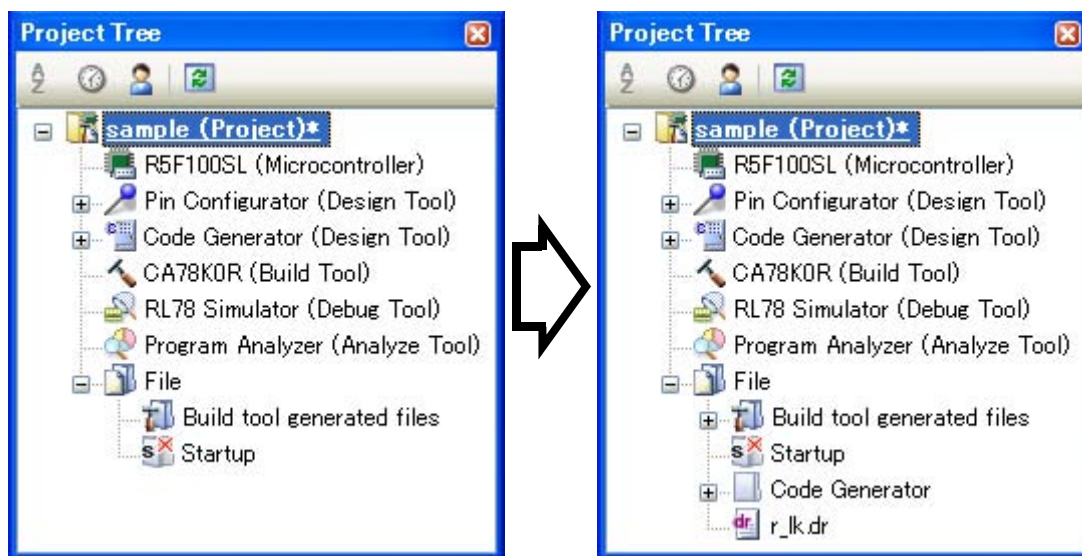
- 3. You cannot edit the source code within the [Code Generator Preview panel](#).
- 4. For some of the API functions (such as API functions for serial array units), values such as the SFR register value are calculated and finalized when the source code is generated (when the  button on the [Code Generator panel](#) is pressed). For this reason, the source code displayed in the [Code Generator Preview panel](#) may not be the same as that would actually be generated.

3.5 Output Source Code

Output the source code (device driver program) by pressing the  button on the [Code Generator panel](#).

The destination folder for the source code is specified by clicking [\[Generation\] tab](#) >> [Output folder] in the [Property panel](#).

Figure 3-5. Output Source Code




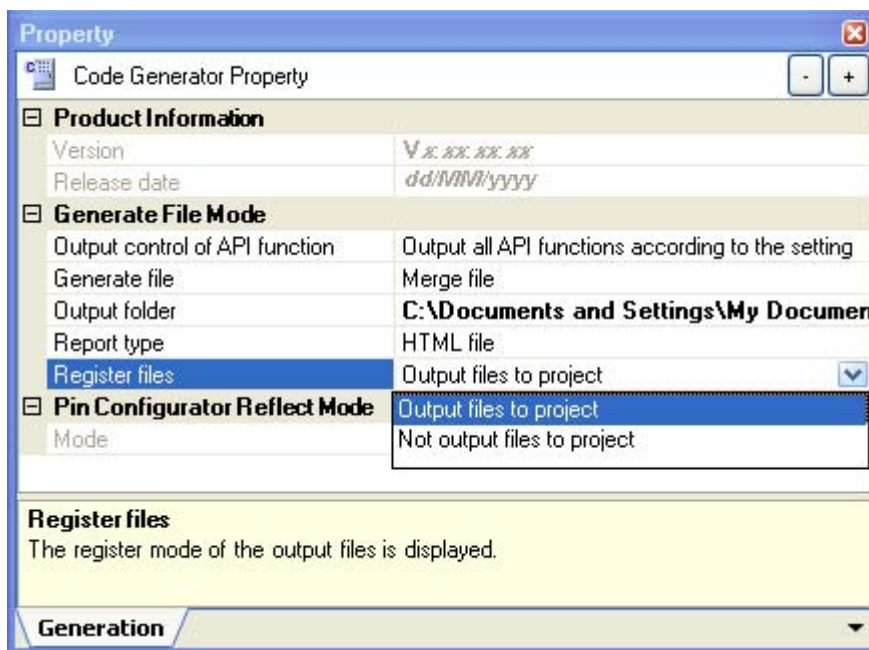
Remark In order to both output source files and add them to the project (display the corresponding source file names in the [Project Tree panel](#)) when you click the  button, you must open the [Property panel](#), and under [\[Generation\] tab](#) >> [Register files], specify "Output files to project".

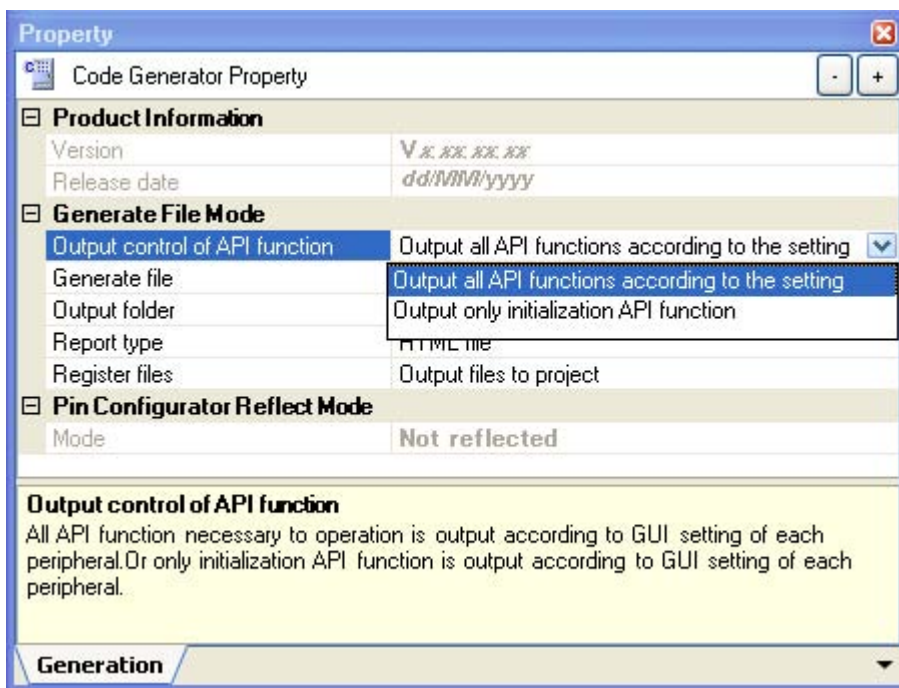
Figure 3-6. Configure Whether to Register



3.5.1 Set whether or not to generate source code

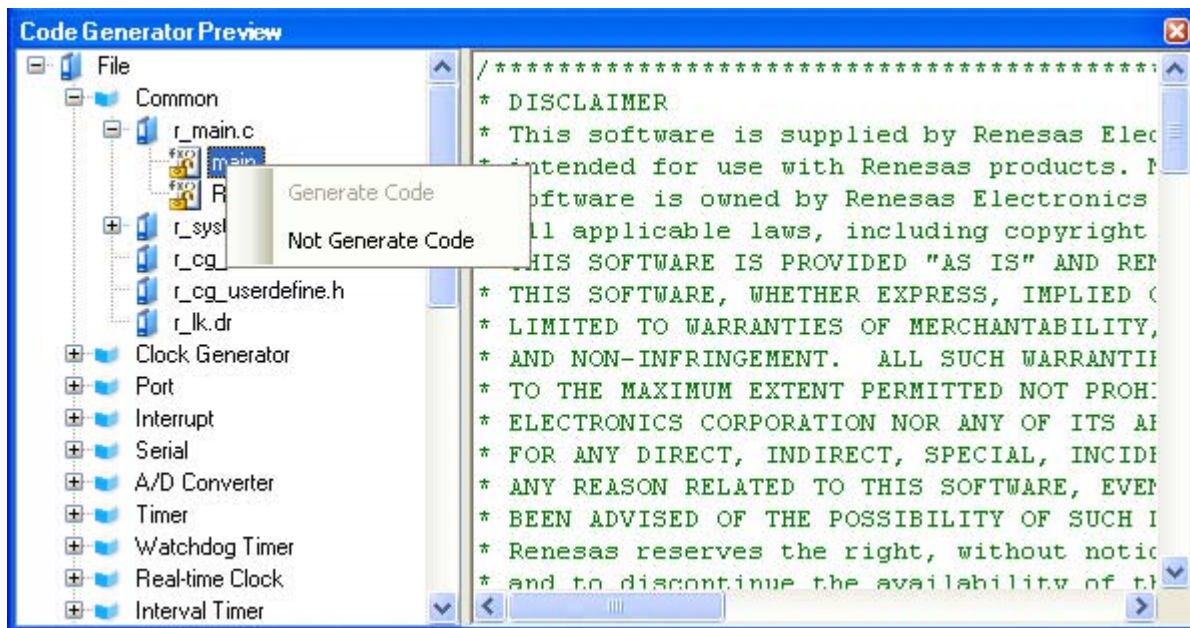
You can set the type of output API functions (all API functions or only initialization API functions) by selecting [Output all API function according to the setting/Output only initialization API function] from [Generation] tab >> [Output control of API function] in the Property panel.

Figure 3-7. Setting That Determines Type of API Functions







You can set whether or not to generate the corresponding source code on a per-API function basis by selecting [Generate code/Not generate code] from the context menu displayed by right clicking the API function name in the Code Generator Preview panel.

Figure 3-8. Setting That Determines Whether or Not to Generate Source Code



Remark You can confirm the current setting for the generation of source code by checking the type of icon in the [Code Generator Preview panel](#).

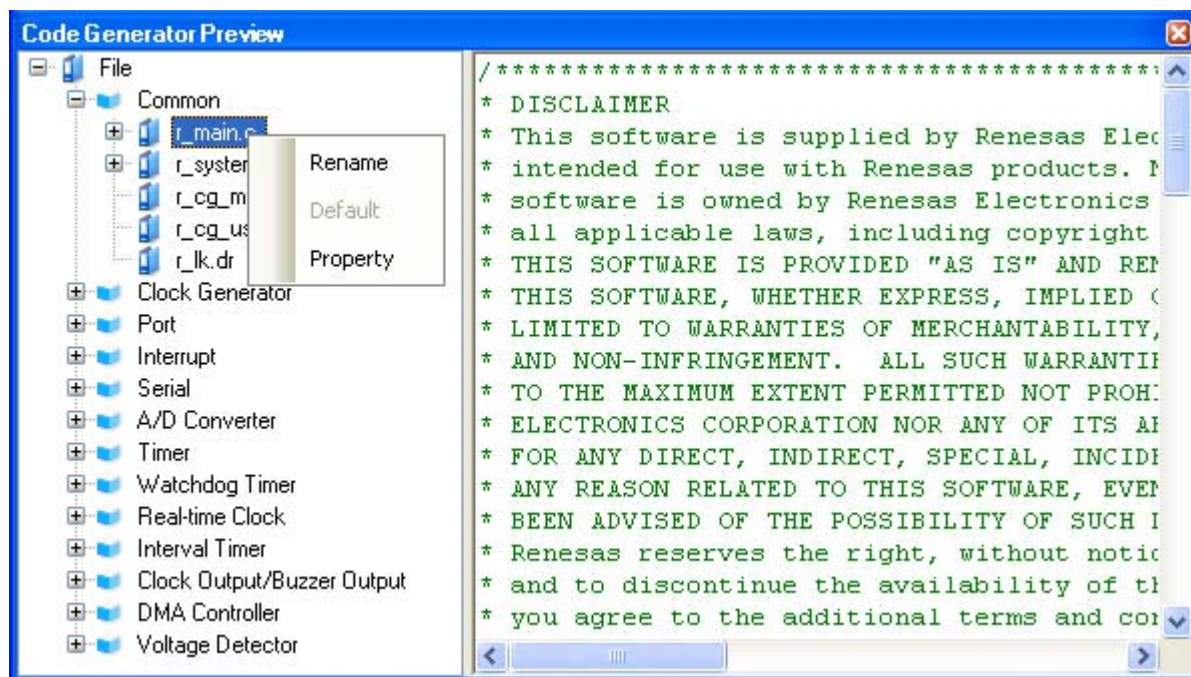
Table 3-4. Setting That Determines Whether or Not to Generate Source Code

Type of Icon	Outline
	Source code for the currently selected API function is generated. If this icon is displayed next to the API function, the corresponding source code must be generated (it is impossible to change the icon to ).
	Source code for the currently selected API function is generated.
	Source code for the currently selected API function is not generated.

3.5.2 Change file name

The Code Generator is used to change the file name by selecting [Rename] from the context menu displayed by right clicking the file name in the [Code Generator Preview panel](#).

Figure 3-9. Change File Name

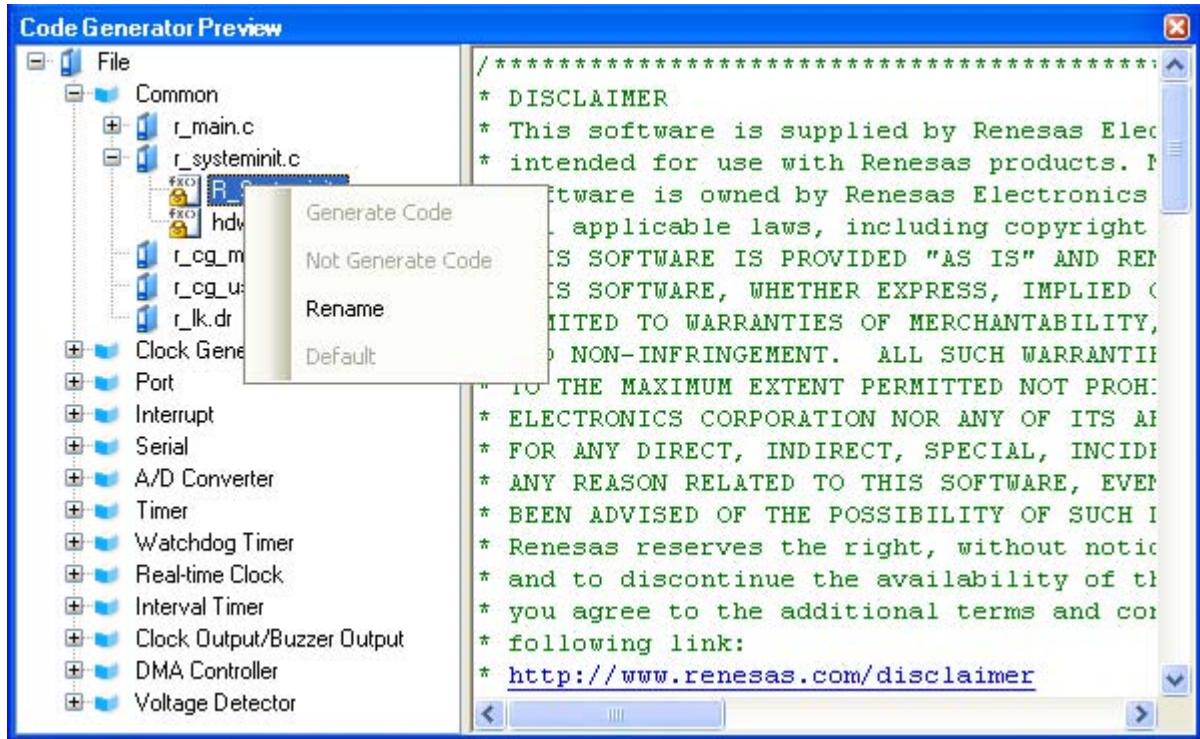


Remark To restore the default file name defined by the Code Generator, select [Default] from the context menu.

3.5.3 Change API function name

The Code Generator is used to change the name of the API function by selecting [Rename] from the context menu displayed by right clicking the API function name in the [Code Generator Preview panel](#).

Figure 3-10. Change API Function Name

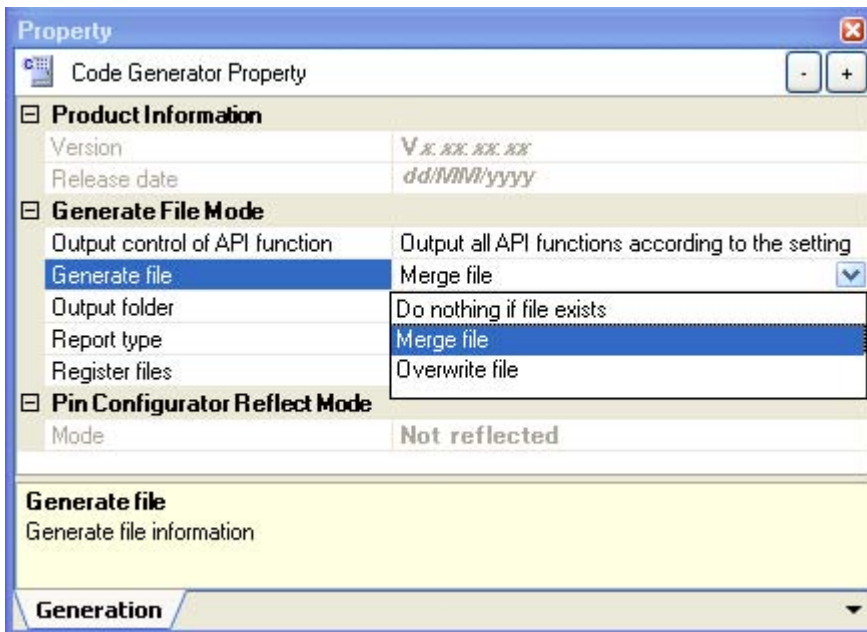


Remark To restore the default name of the API function defined by the Code Generator, select [Default] from the context menu.

3.5.4 Change output mode

The Code Generator is used to change the output mode (Do nothing if file exists, Merge file, Overwrite file) for the source code by selecting [Generation] tab >> [Generate file] in the Property panel.

Figure 3-11. Change Output Mode



Remark The output mode is selected from the following three types.

Table 3-5. Output Mode of Source Code

Output Mode	Outline
Do nothing if file exists	If a file with the same name exists, a new file will not be output.
Merge file	If a file with the same name exists, a new file is merged with the existing file. Only the section between "/* Start user code Do not edit comment generated here */" and "/* End user code. Do not edit comment generated here */" will be merged.
Overwrite file	If a file with the same name exists, the existing file is overwritten by a new file.

3.5.5 Change output destination folder

The Code Generator is used to change the output destination folder for the source code by selecting [\[Generation\] tab](#) >> [\[Output folder\]](#) in the [Property panel](#).

To change the output destination, use the [Browse For Folder dialog box](#) which opens by pressing the [\[...\]](#) button in the [\[Output folder\]](#).

Figure 3-12. Change Output Destination Folder



3.6 Output Report Files

Output report files (a file containing information configured using Code Generator and a file containing information regarding the source code) by first activating the [Code Generator panel](#) or [Code Generator Preview panel](#), then selecting [File] menu >> [Save Code Generator Report].

The destination folder for the report file is specified by clicking [[Generation](#)] tab >> [Output folder] in the [Property panel](#).

Remarks 1. You can only use "macro" or "function" as a name of the report file.

Table 3-6. Output Report Files

File Name	Outline
macro	A file that contains the information configured using Code Generator
function	A file that contains the information regarding the source code

2. The output mode for the report file is fixed to "Overwrite file".

Figure 3-13. Output Example of Report File "macro"

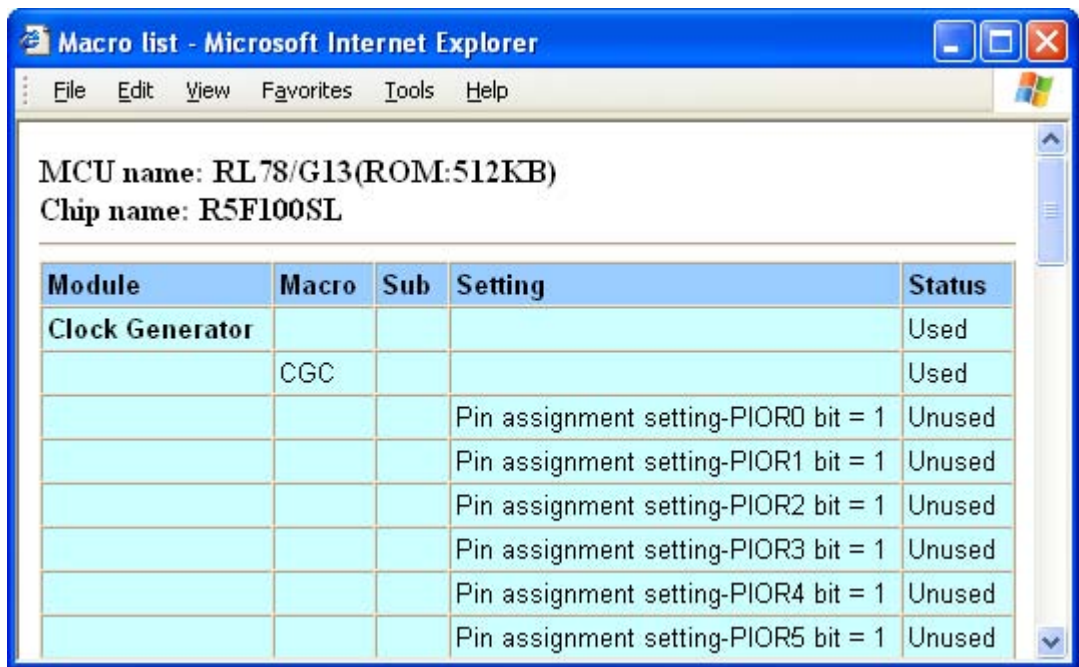


Figure 3-14. Output Example of Report File "function"

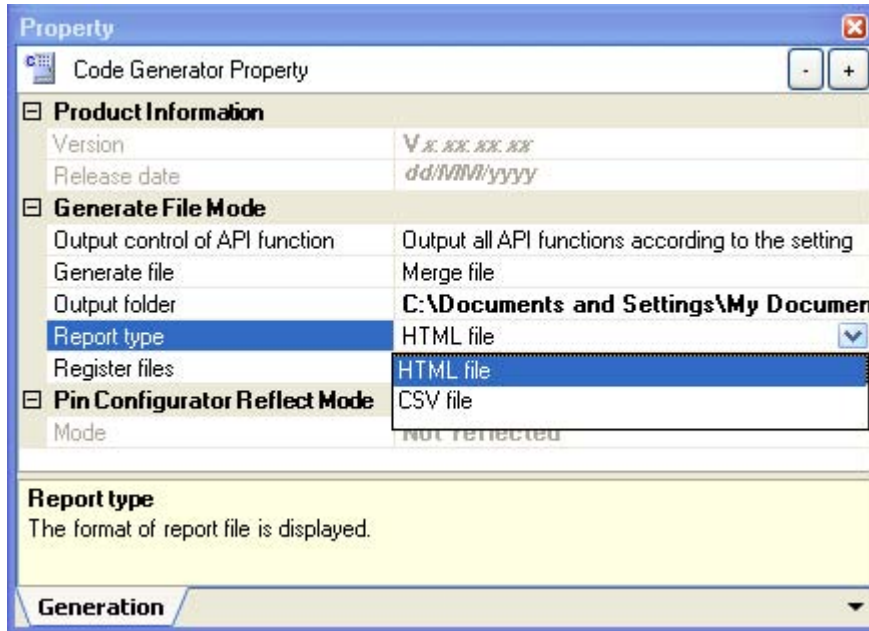
MCU name: RL78/G13(ROM:512KB)
 Chip name: R5F100SL

Module	File	Macro	Function	Default	Status
Common					
	r_main.c			r_main.c	Used
			void main(void)	main	Used
			void R_MAIN_UserInit(void)	R_MAIN_UserInit	Used
	r_systeminit.c			r_systeminit.c	Used
			void R_Systeminit(void)	R_Systeminit	Used
			void hdwinit(void)	hdwinit	Used
	r_cg_macrodriver.h			r_cg_macrodriver.h	Used
	r_cg_userdefine.h			r_cg_userdefine.h	Used
	r_lk.dr			r_lk.dr	Used

3.6.1 Change output format

The Code Generator is used to change the output format (HTML file or CSV file) of the report file by selecting [Generation] tab >> [Report type] in the Property panel.

Figure 3-15. Change Output Format



Remark Output format is selected from the following two types.

Table 3-7. Output Mode of Source Code

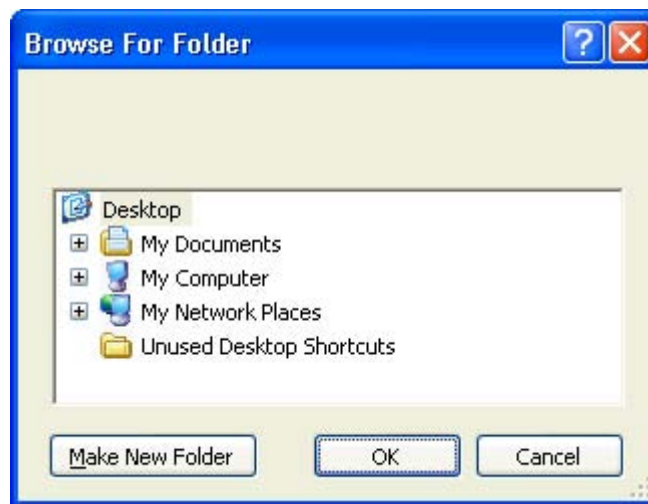
Report Type	Outline
HTML file	Outputs a report file in HTML format.
CSV file	Outputs a report file in CSV format.

3.6.2 Change output destination

The Code Generator is used to change the output destination folder for the report file by selecting [\[Generation\] tab >> \[Output folder\]](#) in the [Property panel](#).

To change the output destination, use the [Browse For Folder dialog box](#) which opens by pressing the [...] button in the [Output folder].

Figure 3-16. Change Output Destination



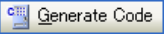
APPENDIX A WINDOW REFERENCE

This appendix explains in detail the functions of the windows, panels and dialog boxes of the design tool.

A.1 Description

The design tool has the following windows, panels and dialog boxes.

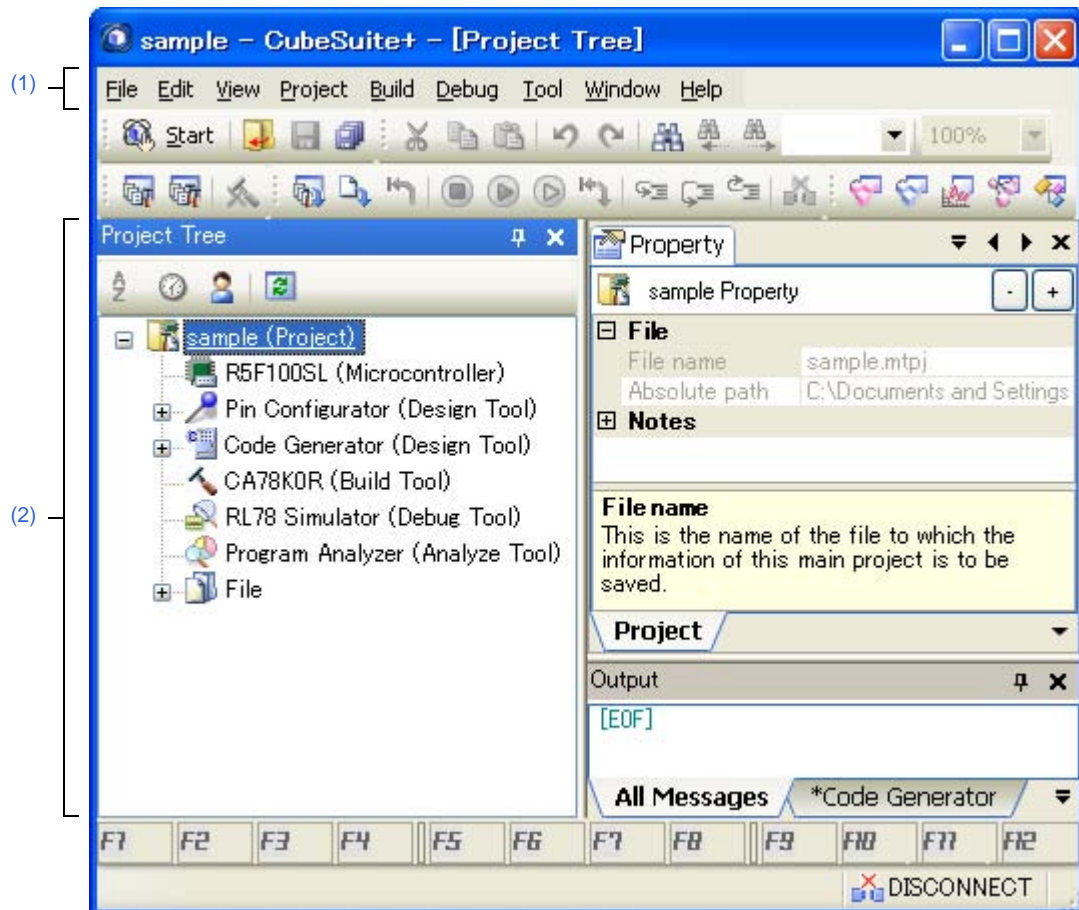
Table A-1. Window/Panel/Dialog Box List

Window/Panel/Dialog Box Name	Function
Main window	This is the first window to open when CubeSuite+ is launched. This window is used to operate various components (design tool, build tool, etc.) provided by CubeSuite+.
Project Tree panel	This panel displays the components of the project (microcontroller, design tool, build tool, etc.) in a tree structure.
Property panel	This panel allows you to view the information and change the setting for the node selected in the Project Tree panel , the peripheral function button pressed in the Code Generator panel or the file selected in the Code Generator Preview panel .
Device Pin List panel	This panel allows you to enter information on each pin of the microcontroller.
Device Top View panel	This panel displays the information entered in the Device Pin List panel .
Code Generator panel	This panel allows you to configure the information necessary to control the peripheral functions provided by the microcontroller.
Code Generator Preview panel	This panel allows you to confirm or configure on a per-API function basis the setting that determines whether or not the source code (device driver program) is generated when the  button is pressed in the Code Generator panel . It also allows you to confirm the source code that reflects the information configured in the Code Generator panel .
Output panel	This panel displays operation logs for various components (design tool, build tool, etc.) provided by CubeSuite+.
Column Chooser dialog box	This dialog box allows you to choose whether or not to display the item listed in this dialog box in the device pin list, and add columns to or delete columns from the device pin list.
New Column dialog box	This dialog box allows you to add your own column to the device pin list.
Browse For Folder dialog box	This dialog box allows you to specify the output destination for files (source code, report file, etc.).
Save As dialog box	This dialog box allows you to name and save a file (such as a report file).

Main window

This is the first window to open when CubeSuite+ is launched. This window is used to operate various components (design tool, build tool, etc.) provided by CubeSuite+.

Figure A-1. Main Window



The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)

[How to open]

- From the [start] menu, select [All Programs] >> [Renesas Electronics CubeSuite+] >> [CubeSuite+].

[Description of each area]

(1) Menu bar

This area consists of the following menu items.

(a) [File] menu

Save Pin List	Device Pin List panel -dedicated item Saves a report file (a file containing information configured using Pin Configurator: device pin list) overwriting the existing file.
Save Pin List As...	Device Pin List panel -dedicated item Opens the Save As dialog box for naming and saving a report file (a file containing information configured using Pin Configurator: device pin list).
Save Top View	Device Top View panel -dedicated item Saves a report file (a file containing information configured using Pin Configurator: device top view) overwriting the existing file.
Save Top View As...	Device Top View panel -dedicated item Opens the Save As dialog box for naming and saving a report file (a file containing information configured using Pin Configurator: device top view).
Save Code Generator Report	Code Generator panel / Code Generator Preview panel -dedicated item Outputs report files (a file containing information configured using Code Generator and a file containing information regarding the source code). - The output format for the report file (either HTML or CSV) is selected by clicking [Generation] tab >> [Report type] in the Property panel . - The destination folder for the report file is specified by clicking [Generation] tab >> [Output folder] in the Property panel .
Save Output- <i>Tab Name</i>	Output panel -dedicated item Saves the message corresponding to the specified tab overwriting the existing file.
Save Output- <i>Tab Name</i> As...	Output panel -dedicated item Opens the Save As dialog box for naming and saving the message corresponding to the specified tab.

(b) [Edit] menu

Undo	Property panel -dedicated item Cancels the effect of an edit operation to restore the previous state.
Cut	Property panel -dedicated item Sends the character string or lines selected with range selection to the clipboard and deletes them.
Copy	Property panel / Output panel -dedicated item Sends the character string or lines selected with range selection to the clipboard.
Paste	Property panel -dedicated item Inserts the contents of the clipboard at the caret position.
Delete	Property panel -dedicated item Deletes the character string or the lines selected with the range selection.
Select All	Property panel / Output panel -dedicated item Selects all the strings displayed in the item being edited or all the strings displayed in the Message area .

Search...	Device Pin List panel/Code Generator Preview panel/Output panel -dedicated item Opens the Search and Replace dialog box for searching strings with the [Quick Search] tab selected.
Replace...	Output panel -dedicated item Opens the Search and Replace dialog box for replacing strings with the [Whole Replace] tab selected.

(c) [Help] menu

Open Help for Project Tree Panel	Project Tree panel -dedicated item Displays the help of Project Tree panel .
Open Help for Property Panel	Property panel -dedicated item Displays the help of Property panel .
Open Help for Device Pin List Panel	Device Pin List panel -dedicated item Displays the help of Device Pin List panel .
Open Help for Device Top View Panel	Device Top View panel -dedicated item Displays the help of Device Top View panel .
Open Help for Code Generator Panel	Code Generator panel -dedicated item Displays the help of Code Generator panel .
Open Help for Code Generator Preview Panel	Code Generator Preview panel -dedicated item Displays the help of Code Generator Preview panel .
Open Help for Output Panel	Output panel -dedicated item Displays the help of Output panel .

(2) Panel display area

This area consists of multiple panels, each dedicated to a different purpose.

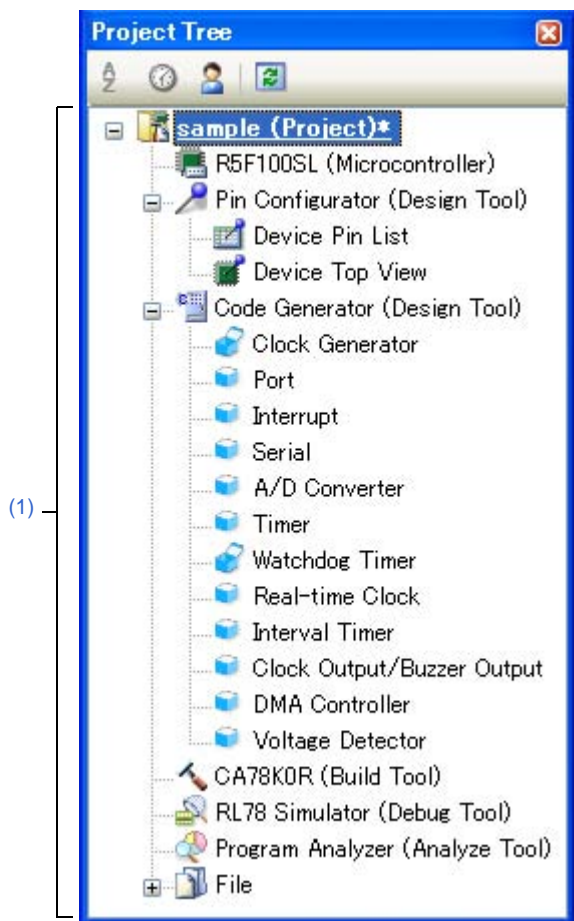
See the following sections for details on this area.

- [Project Tree panel](#)
- [Property panel](#)
- [Device Pin List panel](#)
- [Device Top View panel](#)
- [Code Generator panel](#)
- [Code Generator Preview panel](#)
- [Output panel](#)

Project Tree panel

This panel displays components of the project (microcontroller, design tool, build tool, etc.) in a tree structure.

Figure A-2. Project Tree Panel



The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)
- [\[\[Help\] menu \(Project Tree panel-dedicated items\)\]](#)
- [\[Context menu\]](#)

[How to open]

- From the [View] menu, select [Project Tree].

[Description of each area]

(1) Project tree area

This area displays components of the project (microcontroller, design tool, build tool, etc.) in a tree structure.

(a) Pin Configurator (Design Tool)

This node consists of the following pin nodes.

Device Pin List	Opens the Device Pin List panel for entering information on the pins of the microcontroller.
Device Top View	Opens the Device Top View panel that displays the information entered in the Device Pin List panel .

(b) Code Generator (Design Tool)

This node consists of the following peripheral function nodes.




When there is peripheral function target microcontroller is not supporting, peripheral function button is not displayed.

Clock Generator	Opens the [Clock Generator] for configuring the information necessary to control the functions of clock generator, on-chip debug function and etc. provided by the microcontroller.
Port	Opens the [Port] for configuring the information necessary to control the port functions provided by the microcontroller.
Interrupt	Opens the [Interrupt] for configuring the information necessary to control the interrupt functions and the key interrupt function provided by the microcontroller.
Serial	Opens the [Serial] for configuring the information necessary to control the functions of serial array unit and functions of serial interface provided by the microcontroller.
A/D Converter	Opens the [A/D Converter] for configuring the information necessary to control the function of A/D converter provided by the microcontroller.
D/A Converter	Opens the [D/A Converter] for configuring the information necessary to control the function of D/A converter provided by the microcontroller.
Timer	Opens the [Timer] for configuring the information necessary to control the functions of timer array unit provided by the microcontroller.
Watchdog Timer	Opens the [Watchdog Timer] for configuring the information necessary to control the functions of watchdog timer provided by the microcontroller.
Real-time Clock	Opens the [Real-time Clock] for configuring the information necessary to control the functions of real-time clock provided by the microcontroller.
Interval Timer	Opens the [Interval Timer] for configuring the information necessary to control the functions of interval timer provided by the microcontroller.
Comparator	Opens the [Comparator] for configuring the information necessary to control the functions of comparator provided by the microcontroller.
Clock Output/Buzzer Output	Opens the [Clock Output/Buzzer Output] for configuring the information necessary to control the functions of clock output/buzzer output controller provided by the microcontroller.
Data Transfer Controller	Opens the [Data Transfer Controller] for configuring the information necessary to control the function of data transfer controller provided by the microcontroller.
Event Link Controller	Opens the [Event Link Controller] for configuring the information necessary to control the functions of event link controller provided by the microcontroller.
LCD Controller/Driver	Opens the [LCD Controller/Driver] for configuring the information necessary to control the functions of LCD controller/driver provided by the microcontroller.
Sound Generator	Opens the [Sound Generator] for configuring the information necessary to control the functions of sound generator provided by the microcontroller.

DMA Controller	Opens the [DMA Controller] for configuring the information necessary to control the functions of DMA (Direct Memory Access) controller provided by the microcontroller.
Voltage Detector	Opens the [Voltage Detector] for configuring the information necessary to control the functions of voltage detector provided by the microcontroller.
Programmable Gain Amplifier	Opens the [Programmable Gain Amplifier] for configuring the information necessary to control the functions of programmable gain amplifier provided by the microcontroller.

(c) Icons

The table below displays the meaning of the icon displayed to the left of the string representing the peripheral function node.

	Operation in the corresponding Code Generator panel has been carried out.
	Operation in the corresponding Code Generator panel has not been carried out.
	The problem occurs on the settings became the manipulation to the other peripheral function node influences.

[[Help] menu (Project Tree panel-dedicated items)]

Open Help for Project Tree Panel	Displays the help of this panel.
----------------------------------	----------------------------------

[Context menu]

The following context menu items are displayed by right clicking the mouse.

Return to Reset Value	Restores the information for the selected peripheral function node to its default state.
Property	Opens the Property panel containing the information for the selected node ([Pin Configurator (Design Tool)], [Device Pin List], [Device Top View], [Code Generator (Design Tool)], peripheral function node "Clock Generator, Port, etc.").

Property panel

This panel allows you to view the information on and change the setting for the node selected in the [Project Tree panel](#), the peripheral function button pressed in the [Code Generator panel](#) or the file selected in the [Code Generator Preview panel](#).

Figure A-3. Property Panel (Selected [Pin Configurator (Design Tool)])





The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)
- [\[\[Edit\] menu \(Property panel-dedicated items\)\]](#)
- [\[Context menu\]](#)



[How to open]

- On the [Project Tree panel](#), select a node ([Pin Configurator (Design Tool)], [Device Pin List], [Device Top View], [Code Generator (Design Tool)], peripheral function node "[Clock Generator], [Port], etc."), and then select [Property] from the [View] menu.
- On the [Project Tree panel](#), select a node ([Pin Configurator (Design Tool)], [Device Pin List], [Device Top View], [Code Generator (Design Tool)], peripheral function node "[Clock Generator], [Port], etc."), and then select [Property] from the context menu.
- On the [Code Generator Preview panel](#), select a file, and then select [Property] from the [View] menu.
- On the [Code Generator Preview panel](#), select a file, and then select [Property] from the context menu.



- Remarks 1.** If this panel is already open, selecting a different node ([Pin Configurator (Design Tool)], [Device Pin List], [Device Top View], [Code Generator (Design Tool)] or peripheral function node (such as [Clock Generator], [Port], etc.)) in the [Project Tree panel](#) changes the content displayed in the [Detail information display/change area](#) and [Tab selection area](#) accordingly.
- 2.** If this panel is already open, pressing a different peripheral function button (such as  ,  , etc.) in the [Code Generator panel](#) changes the content displayed in the [Detail information display/change area](#) and [Tab selection area](#) accordingly.
- 3.** If this panel is already open, selecting a different file in the [Code Generator Preview panel](#) changes the content displayed in the [Detail information display/change area](#) and [Tab selection area](#) accordingly.

[Description of each area]

(1) Detail information display/change area

This area allows you to view the information on and change the setting for the node ([Pin Configurator (Design Tool)], [Device Pin List], [Device Top View], [Code Generator (Design Tool)] or peripheral function node (such as [Clock Generator], [Port], etc.)) selected in the [Project Tree panel](#), the peripheral function button (such as  ,  , etc.) pressed in the [Code Generator panel](#), or the file selected in the [Code Generator Preview panel](#). The content displayed in this area differs depending on the node selected in the [Project Tree panel](#), the peripheral function button pressed in the [Code Generator panel](#) or the file selected in the [Code Generator Preview panel](#).

The following table displays the meaning of  and  displayed to the left of each category.

	Indicates that the items within the category are displayed as a "collapsed view".
	Indicates that the items within the category are displayed as an "expanded view".

Remark To switch between  and  , click this mark or double-click the category name.

(2) Tab selection area

Categories for the display of the detailed information are changed when each tab is selected. In this panel, following tabs are contained (see the section explaining each tab for details on the display/setting on the tab).

- [\[Pin Configurator Settings\] tab](#)
- [\[Device Pin List Information\] tab](#)
- [\[Device Top View Settings\] tab](#)
- [\[Generation\] tab](#)
- [\[Macro Setting\] tab](#)
- [\[File Setting\] tab](#)

[[Edit] menu (Property panel-dedicated items)]

Undo	Cancels the effect of an edit operation to restore the previous state.
Cut	Sends the character string or lines selected with range selection to the clipboard and deletes them.
Copy	Sends the character string or lines selected with range selection to the clipboard.
Paste	Inserts the contents of the clipboard at the caret position.
Delete	Deletes the character string or the lines selected with the range selection.
Select All	Selects all strings displayed in the item being edited.

[Context menu]

The following context menu items are displayed by right clicking the mouse.

(1) While the item is being edited

Undo	Cancels the effect of an edit operation to restore the previous state.
Cut	Sends the character string or lines selected with range selection to the clipboard and deletes them.

Copy	Sends the character string or lines selected with range selection to the clipboard.
Paste	Inserts the contents of the clipboard at the caret position.
Delete	Deletes the character string or the lines selected with the range selection.
Select All	Selects all strings displayed in the item being edited.

(2) While the item is not being edited

Property Reset to Default	Restores the selected item to its default state.
Property Reset All to Default	Restores all items to their default state.

[Pin Configurator Settings] tab

This tab displays information (Product Information and Package) on the [Pin Configurator (Design Tool)] selected in the [Project Tree panel](#).

Figure A-4. [Pin Configurator Settings] Tab



The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)

[How to open]

- On the [Project Tree panel](#), select [*Project name* (Project)] >> [Pin Configurator (Design Tool)], and then select [Property] from the [View] menu.
- On the [Project Tree panel](#), select [*Project name* (Project)] >> [Pin Configurator (Design Tool)], and then select [Property] from the context menu.

Remark If this panel is already open, selecting a different [Pin Configurator (Design Tool)] in the [Project Tree panel](#) changes the content displayed accordingly.

[Description of each area]

(1) [Product Information] category

This area displays product information (Version and Release date) on Pin Configurator.

Version	Displays the version of Pin Configurator (Pin Configurator Plug-in).
Release date	Displays the release date of Pin Configurator (Pin Configurator Plug-in).

(2) [Package] category

Change the shape (Package type) and settings of the microcontroller to display as the device top view in the [Device Top View panel](#).

Package type	Selects the shape of the microcontroller displayed in the device top view.
--------------	--

[Device Pin List Information] tab

This tab displays information (Product Information) on the [Device Pin List] selected in the [Project Tree panel](#).

Figure A-5. [Device Pin List Information] Tab



The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)

[How to open]

- On the [Project Tree panel](#), select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List], and then select [Property] from the [View] menu.
- On the [Project Tree panel](#), select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List], and then select [Property] from the context menu.

Remark If this panel is already open, selecting a different [Device Pin List] in the [Project Tree panel](#) changes the content displayed accordingly.

[Description of each area]

(1) [Product Information] category

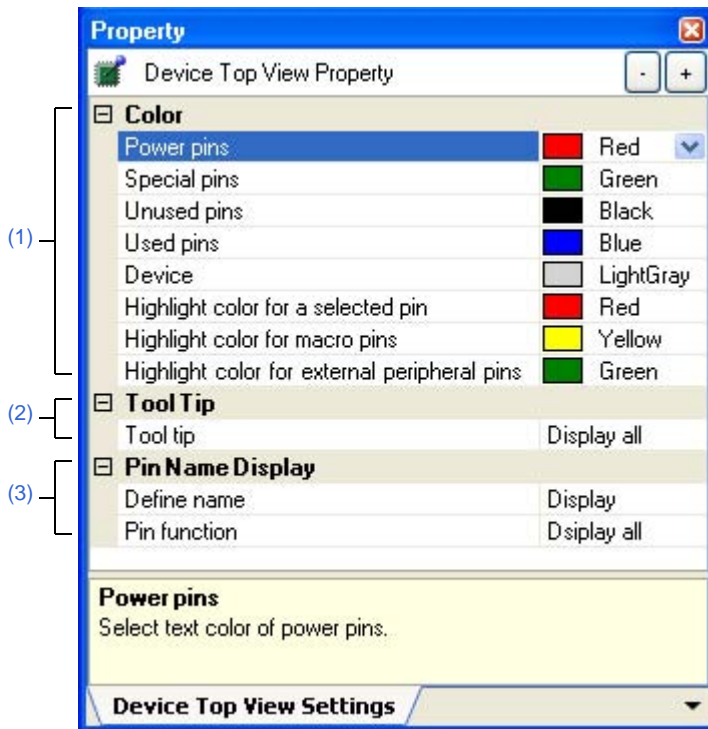
This area displays product information (Version and Release date) on Pin Configurator.

Version	Displays the version of Pin Configurator (Pin Configurator Plug-in).
Release date	Displays the release date of Pin Configurator (Pin Configurator Plug-in).

[Device Top View Settings] tab

This tab allows you to view the information (Color, Tool Tip and Pin Name Display) on and change the setting for the [Device Top View] selected in the [Project Tree panel](#).

Figure A-6. [Device Top View Settings] Tab



The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)

[How to open]

- On the [Project Tree panel](#), select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Top View], and then select [Property] from the [View] menu.
- On the [Project Tree panel](#), select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Top View], and then select [Property] from the context menu.

Remark If this panel is already open, selecting a different [Device Top View] in the [Project Tree panel](#) changes the content displayed accordingly.

[Description of each area]

(1) [Color] category

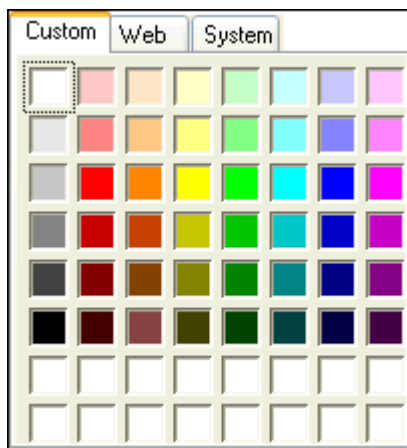
Select the display colors to differentiate the pin groups (Power pins, Special pins, etc.) in the device top view.

Power pins	Selects the display color for power pins (pins whose use is limited to power).
Special pins	Selects the display color for special pins (pins with specified uses).

Unused pins	Selects the display color for unused pins (dual-use pins with no use set in the Device Pin List panel).
Used pins	Selects the display color for used pins (dual-use pins with a use set in the Device Pin List panel).
Device	Selects the display color of the microcontroller.
Highlight color for a selected pin	Selects the background color of a pin selected in the Device Pin List panel , on the [Pin Number] tab .
Highlight color for macro pins	Selects the background color of pins selected in the Device Pin List panel , on the [Macro] tab .
Highlight color for external peripheral pins	Selects the background color of pins selected in the Device Pin List panel , on the [External Peripheral] tab .

Remark To change the setting of the color, use the following color palette which opens by making a selection from the dropdown list in this area.

Figure A-7. Color Palette



(2) [Tool Tip] category

Select whether to display a tooltip with information about a pin when the mouse cursor is moved over the pin in the device top view.

Tool tip	Selects whether to display a tooltip with information about a pin when the mouse cursor is moved over the pin in the device top view panel.	
	Display all	Displays the "Description", "Recommend Connection for Unused", and "Attention" strings for the device pin list.
	Description / recommended connection for unused pin only	Displays the "Description", and "Recommend Connection for Unused" strings for the device pin list.
	Attention only	Displays the "Attention" string for the device pin list.
	Not display	Hides tooltips when the mouse cursor hovers over a pin.

(3) [Pin Name Display] category

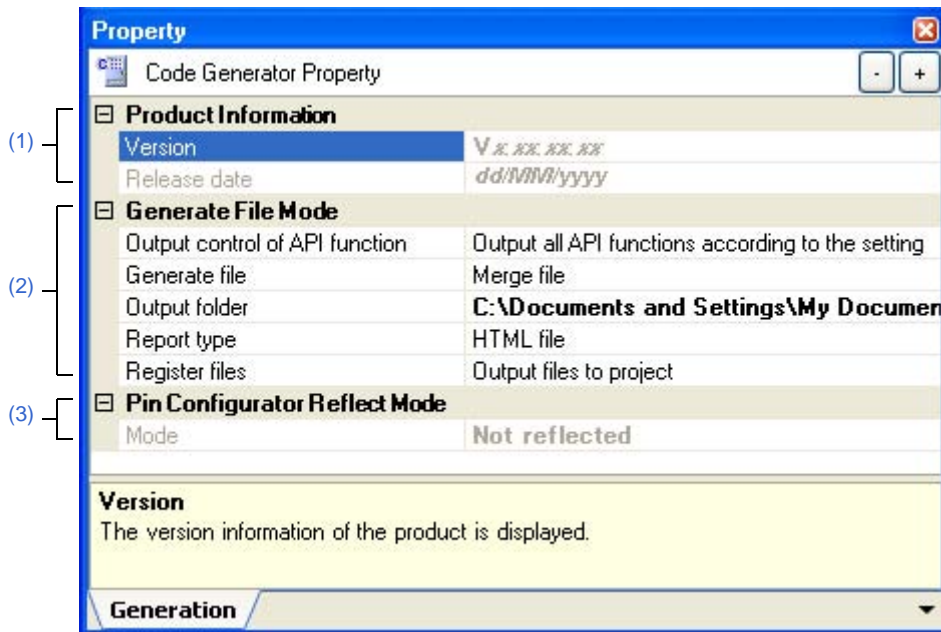
Select whether to display additional information about the pin in the device top view.

Define name	Selects whether to display the "Define Name" string of the device pin list appended to the pin in the device top view.	
	Display	Displays the "Define Name" string of the device pin list in appended format.
	Not display	Hides the "Define Name" string of the device pin list.
Pin function	Selects whether to also display unselected functions in the device top view when a function has been selected from the device pin list's "Function" feature.	
	Display all	Displays functions selected via the device pin list's "Function" feature in parentheses.
	Selected function only	Only display functions selected via the device pin list's "Function" feature in the device top view.

[Generation] tab

This tab allows you to view the information (Product Information, Generate File Mode and Pin Configurator Reflect Mode) on and change the setting for the [Code Generator (Design Tool)] selected in the [Project Tree panel](#).

Figure A-8. [Generation] Tab



The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)

[How to open]

- On the [Project Tree panel](#), select [*Project name* (Project)] >> [Code Generator (Design Tool)], and then select [Property] from the [View] menu.
- On the [Project Tree panel](#), select [*Project name* (Project)] >> [Code Generator (Design Tool)], and then select [Property] from the context menu.

Remark If this panel is already open, selecting a different [Code Generator (Design Tool)] in the [Project Tree panel](#) changes the content displayed accordingly.

[Description of each area]


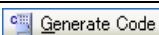

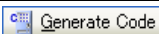
(1) [Product Information] category

This area displays product information (Version and Release date) on Code Generator.

Version	Displays the version of Code Generator (Code Library).
Release date	Displays the release date of Code Generator (Code Library).

(2) [Generate File Mode] category


This area allows you to view and change the setting for the file generation mode (Output control of API function, Generate file, etc.) of Code Generator.

Output control of API function	Views or selects the type of output API functions (all API functions or only initialization API functions) when the  button is pressed.	
	Output all API functions according to the setting	Outputs all API functions.
	Output only initialization API function	Outputs only initialization API functions.
Generate file	Views or selects the operation mode applied when the  button is pressed. Operation mode applied when you select [File] menu >> [Save Code Generator Report] is fixed to "Overwrite file".	
	Do nothing if file exists	If a file with the same name exists, a new file will not be output.
	Merge file	If a file with the same name exists, a new file is merged with the existing file. Only the section between <code>/* Start user code ...</code> . Do not edit comment generated here <code>*/</code> and <code>/* End user code.</code> Do not edit comment generated here <code>*/</code> will be merged.
	Overwrite file	If a file with the same name exists, the existing file is overwritten by a new file.
Output folder	Views or selects the destination folder for various files (source code and report files) which are output when the  button is pressed or when [File] menu >> [Save Code Generator Report] is selected.	
Report type	Views or selects the format of the report files (a file containing information configured using Code Generator and a file containing information regarding the source code) which are output when [File] menu >> [Save Code Generator Report] is selected.	
	HTML file	Outputs a report file in HTML format.
	CSV file	Outputs a report file in CSV format.
Register files	Selects whether source code generated by pressing the  button should be added to the project.	
	Output files to project	Adds output source code to the project. The source code will be added to the Project Tree panel , under the [File] - [Code Generator] node.
	Not output files to project	Does not add output source code to the project.

Remark To change the output destination, use the [Browse For Folder dialog box](#) which opens by pressing the [...] button in this area.

(3) [Pin Configurator Reflect Mode] category

Configure the information linking (Mode) between Code Generator and Pin Configurator.

Mode	Selects whether to reflect the settings made in the Code Generator panel in the Device Pin List panel when the  button is pressed.	
	Reflected	Reflects Code Generator panel settings in the Device Pin List panel .
	Not reflected	Does not reflect Code Generator panel settings in the Device Pin List panel .

Remark If "Not reflected" is selected, then the  button will be grayed out (disabled).

[Macro Setting] tab



This tab allows you to view the information (Macro Information) on and change the setting for the peripheral function node "[Clock Generator], [Port], etc." selected in the [Project Tree panel](#), or the peripheral function button " ,  , etc." pressed in the [Code Generator panel](#).

Figure A-9. [Macro Setting] Tab





The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)

[How to open]

- On the [Project Tree panel](#), select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> Peripheral function node "[Clock Generator], [Port], etc.", and then select [Property] from the [View] menu.
- On the [Project Tree panel](#), select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> Peripheral function node "[Clock Generator], [Port], etc.", and then select [Property] from the context menu.

- Remarks 1.** If this panel is already open, selecting a different peripheral function node "[Clock Generator], [Port], etc." in the [Project Tree panel](#) changes the content displayed accordingly.
- 2.** If this panel is already open, pressing a different type of peripheral function button " ,  , etc." in the [Code Generator panel](#) changes the content displayed accordingly.

[Description of each area]

(1) [Macro Information] category

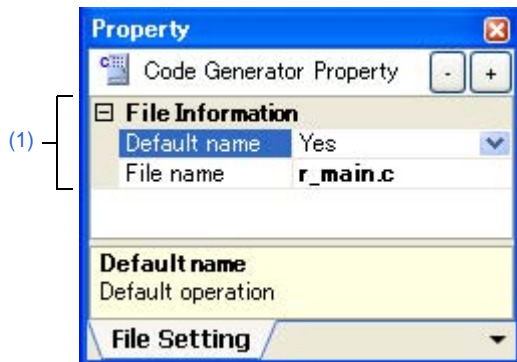
This area allows you to view the information (Macro name) on and change the setting for the peripheral function node "[Clock Generator], [Port], etc." selected in the [Project Tree panel](#), or the peripheral function button pressed in the [Code Generator panel](#).

Macro name	Displays the type of peripheral function node selected in the Project Tree panel or the type of peripheral function button pressed in the Code Generator panel .
------------	--

[File Setting] tab

This tab allows you to view the information (File Information) on and change the setting for the file selected in the [Code Generator Preview panel](#).

Figure A-10. [File Setting] Tab



The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)

[How to open]

- On the [Code Generator Preview panel](#), select a file, and then select [Property] from the [View] menu.
- On the [Code Generator Preview panel](#), select a file, and then select [Property] from the context menu.

Remark If this panel is already open, selecting a different file in the [Code Generator Preview panel](#) changes the content displayed accordingly.

[Description of each area]

(1) [File Information] category

This area allows you to view the information (Default name and File name) on and change the setting for the file selected in the [Code Generator Preview panel](#).

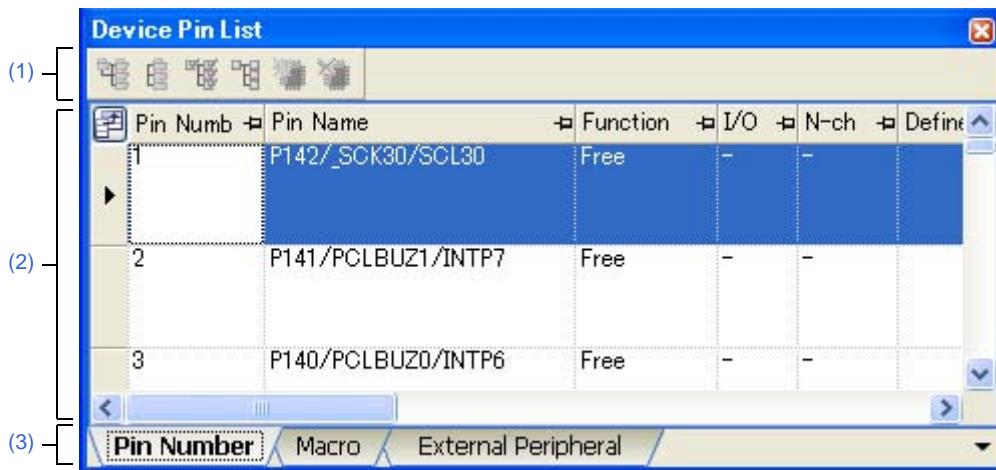
Default name	Views or selects the setting that determines whether the name of the file selected in the Code Generator Preview panel is a default name or not.	
	Yes	The file name is a default name. Changing this area from "No" to "Yes" changes the name of the file to its default name.
	No	The file name is not a default name.
File name	Displays or change the name of the file selected on the Code Generator Preview panel .	

Device Pin List panel

This panel allows you to enter information on each pin of the microcontroller.

Remark The [Device pin list area](#) can be zoomed in and out by in the tool bar, or by operating the mouse wheel while holding down the [Ctrl] key.

Figure A-11. Device Pin List Panel



The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)
- [\[\[File\] menu \(Device Pin List panel-dedicated items\)\]](#)
- [\[\[Help\] menu \(Device Pin List panel-dedicated items\)\]](#)

[How to open]




- On the [Project Tree panel](#), double-click [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List].
- On the [Project Tree panel](#), select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List], and then press the [Enter] key.
- From the [View] menu, select [Pin Configurator] >> [Device Pin List].



[Description of each area]

(1) Toolbar

This area consists of the following buttons.

	Displays the information in the Device pin list area in an expanded view.
	Displays the information in the Device pin list area in a folded view only.
	Clicks this button to automatically process the configuration information in the selected function, I/O, N-ch, and other fields after selecting one of the peripheral functions displayed in the first level on the [Macro] tab .

	Clicks this button to initialize the selected function, I/O, N-ch, and other fields after selecting one of the peripheral functions displayed in the first level on the [Macro] tab .
	Clicks this button to create an external peripheral controller from the external peripheral controller information on the [External Peripheral] tab , and display it in the Device Top View panel .
	Clicks this button to delete the information for the external peripheral controller displayed on the [External Peripheral] tab , on the first layer.

- Remarks 1.** Click the  button to add the information in question as a choice in the "External Parts" column of the [\[Macro\] tab](#) and the [\[Pin Number\] tab](#).
- 2.** Click the  button to remove the external peripheral component in question from the [Device top view area](#) if the [Device Top View panel](#).

(2) Device pin list area

This area displays the "device pin list" for entering information on the pins of the microcontroller.

(3) Tab selection area

Selecting the tab changes the order in which "information on each pin of the microcontroller" is displayed.

This panel has the following tabs:

- [\[Pin Number\] tab](#)
This tab displays information on each pin of the microcontroller in the order of pin number.
- [\[Macro\] tab](#)
This tab displays information on each pin of the microcontroller in the order it was grouped into peripheral functions.
- [\[External Peripheral\] tab](#)
This tab displays information about the pins connected to external peripherals in order grouped at the external-peripheral component level.

[[File] menu (Device Pin List panel-dedicated items)]

Save Pin List	Saves a report file (a file containing information configured using Pin Configurator: device pin list) overwriting the existing file.
Save Pin List As...	Opens the Save As dialog box for naming and saving a report file (a file containing information configured using Pin Configurator: device pin list).

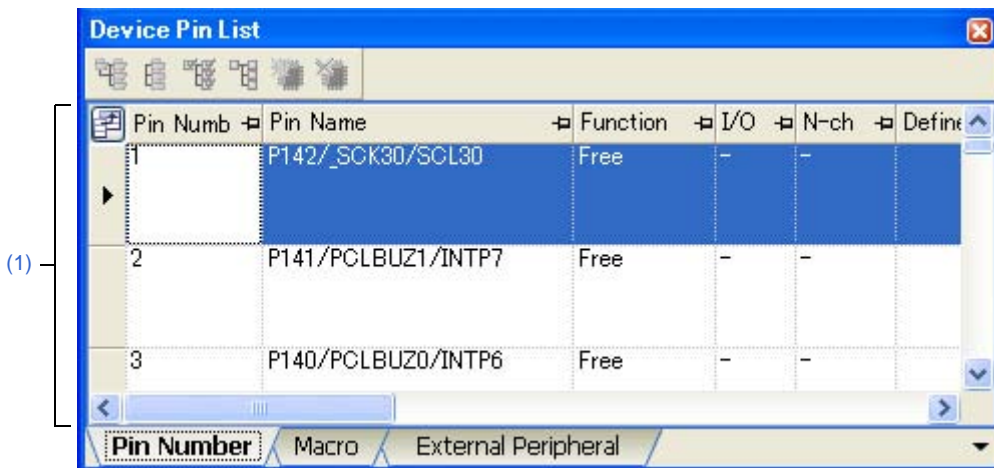
[[Help] menu (Device Pin List panel-dedicated items)]

Open Help for Device Pin List Panel	Displays the help of this panel.
-------------------------------------	----------------------------------

[Pin Number] tab

This tab displays information on each pin of the microcontroller in the order of pin number.

Figure A-12. [Pin Number] Tab



The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)

[How to open]

- On the [Project Tree](#) panel, double-click [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List].
- On the [Project Tree](#) panel, select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List], and then press the [Enter] key.
- From the [View] menu, select [Pin Configurator] >> [Device Pin List].


[Description of each area]

(1) Device pin list area

This area displays the "device pin list" for entering information on the pins of the microcontroller. The device pin list in this area is organized in the order of pin number. The following are the columns comprising the device pin list.

Column Heading	Outline
Pin Number	Displays the pin number of the pin.
Pin Name	This area allows you to select "which function to use" when the pin has more than one functions.
Function	This area allows you to select "which function to use" when the pin has more than one functions.
I/O	This area allows you to select the I/O mode of the pin.
N-ch	This area allows you to select "which output mode to apply" when using the pin in the output mode.

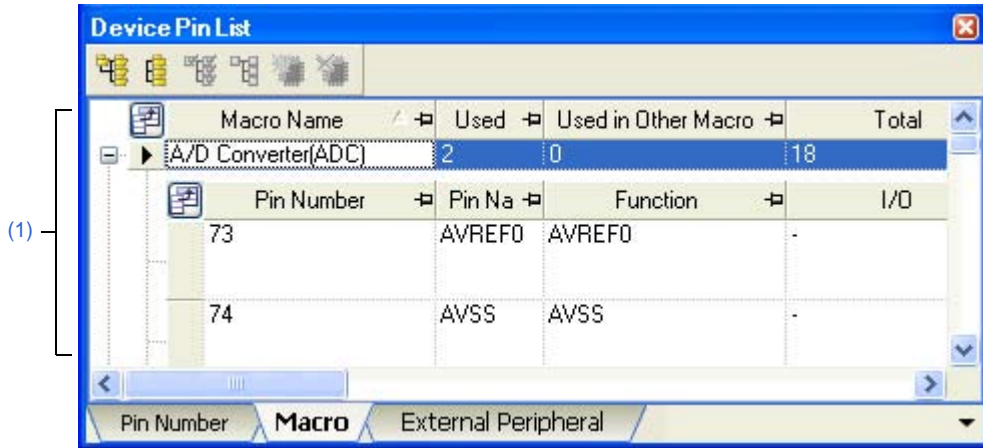
Column Heading	Outline
Define Name	This area allows you to assign a "user-defined pin name" to the pin. wITHIN 256 characters can be entered in the [Define Name].
Description	Displays the summary of function of the pin.
Recommend Connection for Unused	Displays instructions on how to handle the pin when it is not used. This column displays information only when the "Free" is selected in the "Function" column.
Attention	Displays the precaution on using the pin.
External Parts	This area is for selecting which external peripheral controller to connect the pin to.

- Remarks 1.** You cannot add information in the "Pin Number" column, "Pin Name" column, "Description" column, "Recommend Connection for Unused" column and "Attention" column because they contain fixed information.
2. If the "Free" in the "Function" column is changed to a specific pin name, color of the corresponding pin in the [Device Top View panel](#) changes from the "color representing the unused pins" to the "color representing the used pins" selected by clicking [\[Device Top View Settings\] tab](#) >> [Color] in the [Property panel](#).
 3. To move columns (change the display order) in the device pin list, drag and drop the desired column to the desired location.
 4. To add the "user's own column", use the [New Column dialog box](#) which opens by pressing the [New Column...] button in the [Column Chooser dialog box](#) which opens by pressing the  button in the upper left corner of the device pin list.

[Macro] tab

This tab displays information on each pin of the microcontroller in the order it was grouped into peripheral functions.

Figure A-13. [Macro] Tab



The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)

[How to open]

- On the [Project Tree panel](#), double-click [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List].
- On the [Project Tree panel](#), select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List], and then press the [Enter] key.
- From the [View] menu, select [Pin Configurator] >> [Device Pin List].

[Description of each area]

(1) Device pin list area

This area displays the "device pin list" for entering information on the pins of the microcontroller. The device pin list in this area is organized in the order the pins were grouped into peripheral functions.


(a) First layer

The following are the columns comprising the device pin list.

Column Heading	Outline
Macro Name	Displays the name of the peripheral function.
Total	Displays the total number of pins assigned to the peripheral function.
Used	Displays the total number of pins for which the purpose has been set.
Used in Other Macro	Displays the total number of pins for which the purpose has been set by other peripheral functions.

(b) Second layer

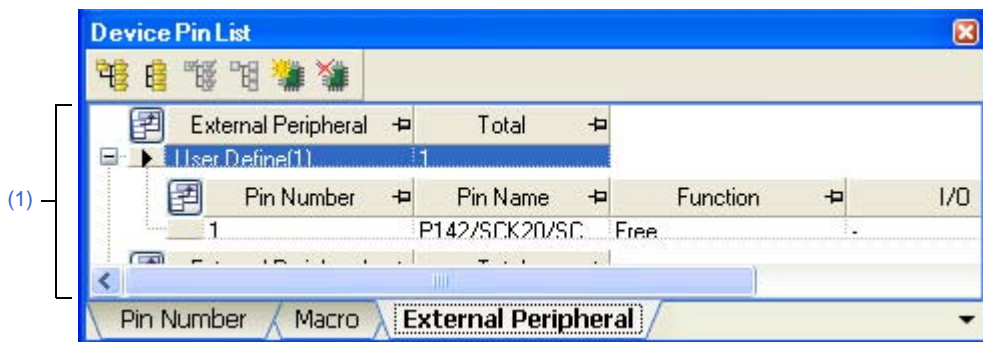
Column Heading	Outline
Pin Number	Displays the pin number of the pin.
Pin Name	Displays the pin name of the pin.
Function	This area allows you to select "which function to use" when the pin has more than one functions.
I/O	This area allows you to select the I/O mode of the pin.
N-ch	This area allows you to select "which output mode to apply" when using the pin in the output mode.
Define Name	This area allows you to assign a "user-defined pin name" to the pin. Within 256 characters can be entered in the [Define Name].
Description	Displays the summary of function of the pin.
Recommend Connection for Unused	Displays instructions on how to handle the pin when it is not used. This column displays information only when the "Free" is selected in the "Function" column.
Attention	Displays the precaution on using the pin.
External Parts	This area is for selecting which external peripheral controller to connect the pin to.

- Remarks 1.** You cannot add information in the "Macro Name", "Total", "Used", "Used by other function", "Pin Number", "Pin Name", "Description", "Recommend Connection for Unused" and "Attention" columns because they contain fixed information.
- If the "Free" in the "Function" column is changed to a specific pin name, color of the corresponding pin in the [Device Top View panel](#) changes from the "color representing the unused pins" to the "color representing the used pins" selected by clicking [\[Device Top View Settings\] tab](#) >> [Color] in the [Property panel](#).
 - To move columns (change the display order) in the device pin list, drag and drop the desired column to the desired location.
 - To add the "user's own column", use the [New Column dialog box](#) which opens by pressing the [New Column...] button in the [Column Chooser dialog box](#) which opens by pressing the  button in the upper left corner of the device pin list.

[External Peripheral] tab

This tab displays information about the pins connected to external peripherals in order grouped at the external-peripheral component level.

Figure A-14. [External Peripheral] Tab



The following items are explained here.

- [How to open]
- [Description of each area]

[How to open]

- On the **Project Tree panel**, double-click [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List].
- On the **Project Tree panel**, select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List], and then press the [Enter] key.
- From the [View] menu, select [Pin Configurator] >> [Device Pin List].

[Description of each area]

(1) Device pin list area

This area displays the "device pin list" for entering information on the pins connected to external peripheral parts. Note that items in this area's device pin list are sorted by groups at the external peripheral controller level.

(a) First layer


The following are the columns comprising the device pin list.

Column Heading	Outline
External Peripheral	Displays the name of the external peripheral controller. To change the name, select this field and then press the [F2] key.
Total	Displays the total number of pins allocated for connection with the microcontroller.

(b) Second layer

Column Heading	Outline
Pin Number	Displays the pin number of the pin.

Column Heading	Outline
Pin Name	Displays the pin name of the pin.
Function	This area allows you to select "which function to use" when the pin has more than one functions.
I/O	This area allows you to select the I/O mode of the pin.
N-ch	This area allows you to select "which output mode to apply" when using the pin in the output mode.
Define Name	This area allows you to assign a "user-defined pin name" to the pin. Within 256 characters can be entered in the [Define Name].
Description	Displays the summary of function of the pin.
Recommend Connection for Unused	Displays instructions on how to handle the pin when it is not used. This column displays information only when the "Free" is selected in the "Function" column.
Attention	Displays the precaution on using the pin.

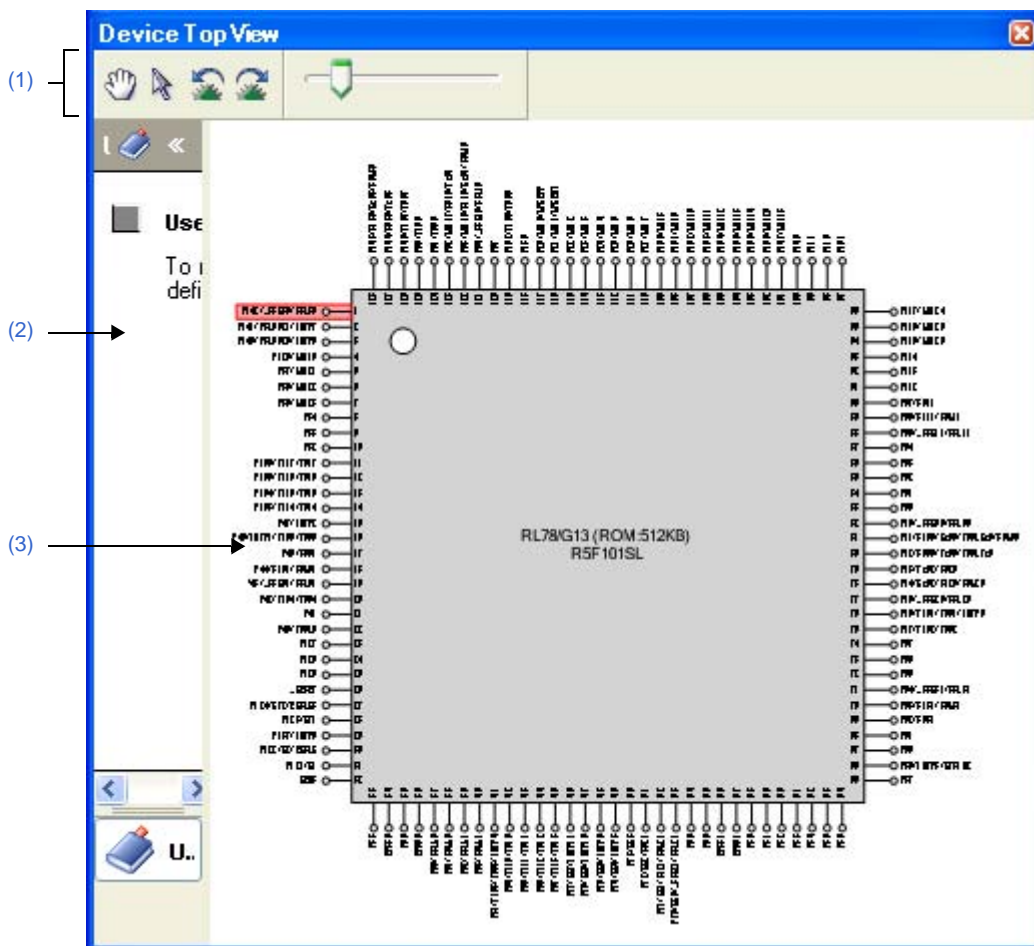
- Remarks 1.** You cannot add information in the "External Peripheral Name", "Connected Pins", "Pin Number", "Pin Name", "Description", "Recommend Connection for Unused" and "Attention" columns because they contain fixed information.
2. If the "Free" in the "Function" column is changed to a specific pin name, color of the corresponding pin in the [Device Top View panel](#) changes from the "color representing the unused pins" to the "color representing the used pins" selected by clicking [\[Device Top View Settings\] tab](#) >> [Color] in the [Property panel](#).
 3. To move columns (change the display order) in the device pin list, drag and drop the desired column to the desired location.
 4. To add the "user's own column", use the [New Column dialog box](#) which opens by pressing the [New Column...] button in the [Column Chooser dialog box](#) which opens by pressing the  button in the upper left corner of the device pin list.

Device Top View panel

This panel displays the information entered in the [Device Pin List panel](#).

Remark The [Device top view area](#) can be zoomed in and out by in the tool bar.

Figure A-15. Device Top View Panel



The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)
- [\[\[File\] menu \(Device Top View panel-dedicated items\)\]](#)
- [\[\[Help\] menu \(Device Top View panel-dedicated items\)\]](#)
- [\[Context menu\]](#)

[How to open]







- On the [Project Tree panel](#), double-click [*Project name* (Project)] >> [[Pin Configurator \(Design Tool\)](#)] >> [[Device Top View](#)].
- On the [Project Tree panel](#), select [*Project name* (Project)] >> [[Pin Configurator \(Design Tool\)](#)] >> [[Device Top View](#)], and then press the [Enter] key.
- From the [View] menu, select [[Pin Configurator](#)] >> [[Device Top View](#)].

Remark In the [Property panel](#), on the [\[Pin Configurator Settings\] tab](#), if "BGA" is selected for the Package type, then this panel cannot be opened.


[Description of each area]

(1) Toolbar

This area consists of the following buttons.

	Clicks this button to enable changing of the display in the Device top view area by drag and drop. By pressing this button, the shape of the mouse cursor in the Device top view area changes from the arrow to the hand.
	Clicks this button to enable moving external peripheral components in the Device top view area to arbitrary locations, and select pins. By pressing this button, the shape of the mouse cursor which has changed into the hand by pressing the  button reverts back to the arrow.
	Rotates the content in the Device top view area 90 degrees counter-clockwise.
	Rotates the content in the Device top view area 90 degrees clockwise.
	Expands or reduces the content in the Device top view area .

(2) [User Define] area

Drag and drop the  button from this area to the [Device top view area](#) to creat and display an external peripheral controller.

(3) Device top view area

This area displays the pin assignment of the microcontroller.
Settings of the pin assignment are displayed using the colors specified by selecting [\[Device Top View Settings\] tab](#) >> [\[Color\]](#) in the [Property panel](#).

Remark If the pin name in the diagram is double-clicked, the [Device Pin List panel](#) opens and the focus moves to the clicked pin in the list.

[[File] menu (Device Top View panel-dedicated items)]

Save Top View	Saves a report file (a file containing information configured using Pin Configurator: device top view) overwriting the existing file.
Save Top View As...	Opens the Save As dialog box for naming and saving a report file (a file containing information configured using Pin Configurator: device top view).

[[Help] menu (Device Top View panel-dedicated items)]

Open Help for Device Top View Panel	Displays the help of this panel.
-------------------------------------	----------------------------------

[Context menu]

When you right click on a pin or external peripheral controller in the [Device top view area](#), the following context menu displays.

(1) When a pin is right clicked

Use as	If the pin has multiple functions, select which function to use.
Connect to External Peripheral	Selects which external peripheral controller to connect the pin to.

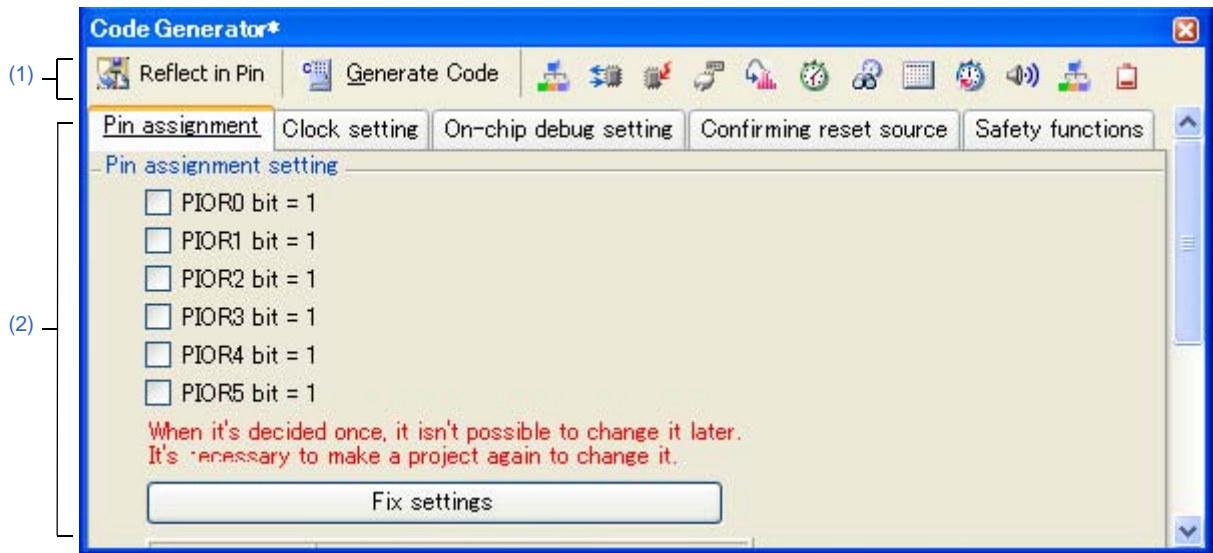
(2) When an external peripheral controller is right clicked

Disconnect Pin	Disconnects from the pin.
Delete External Peripheral	Removes the external peripheral controller.

Code Generator panel

This panel allows you to configure the information necessary to control the peripheral functions provided by the microcontroller.

Figure A-16. Code Generator Panel: [Clock Generator]



The following items are explained here.

- [How to open]
- [Description of each area]
- [[File] menu (Code Generator panel-dedicated items)]
- [[Help] menu (Code Generator panel-dedicated items)]

[How to open]

- On the **Project Tree panel**, double-click [*Project name* (Project)] >> [Code Generator (Design Tool)] >> Peripheral function node "[Clock Generator], [Port], etc.".
- On the **Project Tree panel**, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> Peripheral function node "[Clock Generator], [Port], etc.", and then press the [Enter] key.

Remark If this panel is already open, pressing a different peripheral function button " , , etc." changes the content displayed in the **Information setting area** accordingly.

[Description of each area]





(1) Toolbar

This area consists of the following "peripheral function buttons".

When there is peripheral function target microcontroller is not supporting, peripheral function button is not displayed.

	Reflects settings made on this panel in the Device Pin List panel , and then output the changed contents to the Output panel . This button will be grayed out (disabled) if "Not reflected" is selected in the [Pin Configurator Reflect Mode] category of the [Generation] tab .
--	---

 Generate Code	Outputs the source code (device driver program) to the folder specified by selecting [Generation] tab >> [Output folder] in the Property panel.
	Changes the content displayed in the Information setting area to the "[Clock Generator] for configuring the information necessary to control the functions of clock generator, on-chip debug function and etc. provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Port] for configuring the information necessary to control the port functions provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Interrupt] for configuring the information necessary to control the interrupt functions and the key interrupt function provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Serial] for configuring the information necessary to control the functions of serial array unit and functions of serial interface provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[A/D Converter] for configuring the information necessary to control the function of A/D converter provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[D/A Converter] for configuring the information necessary to control the function of D/A converter provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Timer] for configuring the information necessary to control the functions of timer array unit provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Watchdog Timer] for configuring the information necessary to control the functions of watchdog timer provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Real-time Clock] for configuring the information necessary to control the functions of real-time clock provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Interval Timer] for configuring the information necessary to control the functions of interval timer provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Comparator] for configuring the information necessary to control the functions of comparator provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Clock Output/Buzzer Output] for configuring the information necessary to control the functions of clock output/buzzer output controller provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Data Transfer Controller] for configuring the information necessary to control the function of data transfer controller provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Event Link Controller] for configuring the information necessary to control the functions of event link controller provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[LCD Controller/Driver] for configuring the information necessary to control the functions of LCD controller/driver provided by the microcontroller".

	Changes the content displayed in the Information setting area to the "[Sound Generator] for configuring the information necessary to control the functions of sound generator provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[DMA Controller] for configuring the information necessary to control the functions of DMA (Direct Memory Access) controller provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Voltage Detector] for configuring the information necessary to control the functions of voltage detector provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Programmable Gain Amplifier] for configuring the information necessary to control the functions of programmable gain amplifier provided by the microcontroller".

(2) Information setting area

The content displayed in this area differs depending on the "peripheral function node" or "peripheral function button" selected or pressed when opening this panel.

See user's manual for microcontroller for details on the items to be set.

[[File] menu (Code Generator panel-dedicated items)]

Save Code Generator Report	Outputs report files (a file containing information configured using Code Generator and a file containing information regarding the source code).
----------------------------	---

- Remarks 1.** The output format for the report file (either HTML or CSV) is selected by clicking [\[Generation\] tab](#) >> [\[Report type\]](#) in the [Property panel](#).
- 2.** The destination folder for the report file is specified by clicking [\[Generation\] tab](#) >> [\[Output folder\]](#) in the [Property panel](#).

[[Help] menu (Code Generator panel-dedicated items)]

Open Help for Code Generator Panel	Displays the help of this panel.
------------------------------------	----------------------------------

Code Generator Preview panel


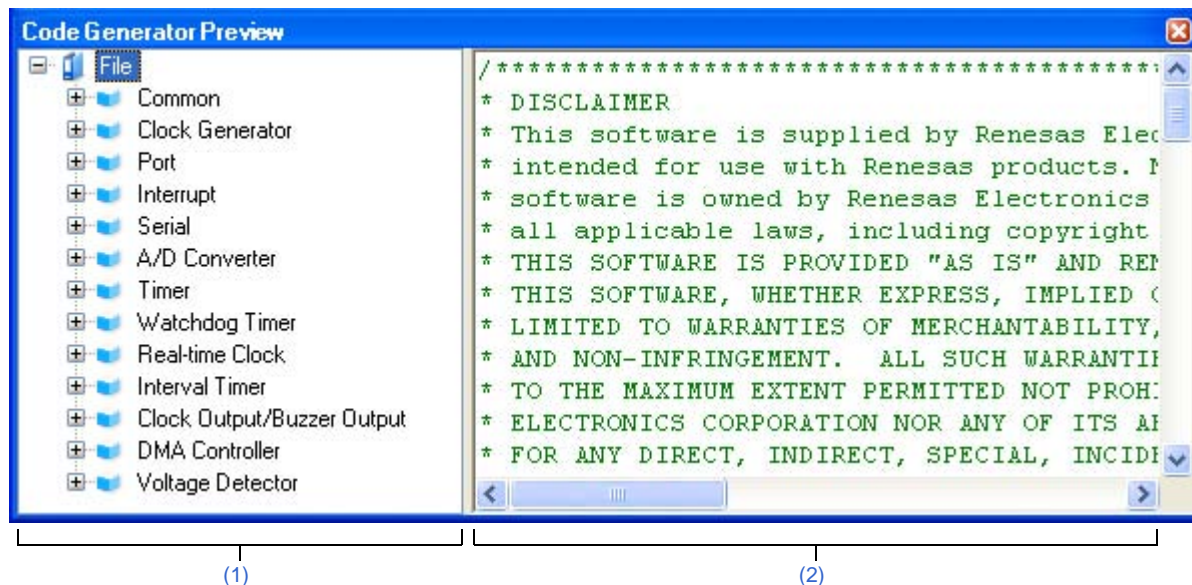
This panel allows you to confirm or configure on a per-API function basis the setting that determines whether or not the source code (device driver program) is generated when the  button is pressed in the [Code Generator panel](#). It also allows you to confirm the source code that reflects the information configured in the [Code Generator panel](#).

Figure A-17. Code Generator Preview Panel



The following items are explained here.


- [\[How to open\]](#)
- [\[Description of each area\]](#)
- [\[\[File\] menu \(Code Generator Preview panel-dedicated items\)\]](#)
- [\[\[Help\] menu \(Code Generator Preview panel-dedicated items\)\]](#)
- [\[Context menu\]](#)

[How to open]

- From the [View] menu, select [Code Generator Preview].

[Description of each area]





(1) Preview tree

This area allows you to confirm or configure on a per-API function basis the setting that determines whether or not the source code (device driver program) is generated when the  button is pressed in the [Code Generator panel](#).

- Remarks 1.** You can change the source code to be displayed by selecting the source file name or API function name in this tree.
- 2.** To select whether or not to generate the source code, use the context menu (Generate code/Not generate code) which is displayed by right-clicking the mouse while the mouse cursor is on the desired icon in the tree.

3. You can confirm the current setting that determines whether or not to generate the source code by checking the type of icon.

Table A-2. Setting That Determines Whether or Not to Generate the Source Code

Type of Icon	Outline
	Source code for the currently selected API function is generated. If this icon is displayed next to the API function, the corresponding source code must be generated (it is impossible to change the icon to ).
	Source code for the currently selected API function is generated.
	Source code for the currently selected API function is not generated.


(2) Source code display area

This area allows you to confirm the source code (device driver program) that reflects the information configured in the [Code Generator panel](#).

The following table displays the meaning of the color of the source code text displayed in this area.

Table A-3. Color of Source Code

Color	Outline
Green	Comment
Blue	Reserved word for C compiler
Red	Numeric value
Black	Code section
Gray	File name

- Remarks 1.** You cannot edit the source code within this panel.
2. For some of the API functions (such as API functions for serial array units), values such as the SFR register value are calculated and finalized when the source code is generated (when the  **Generate Code** button on the [Code Generator panel](#) is pressed). For this reason, the source code displayed in this panel may not be the same as that would actually be generated.
 3. You can change the source code to be displayed by selecting the source file name or API function name in the preview tree.

[[File] menu (Code Generator Preview panel-dedicated items)]

Save Code Generator Report	Outputs report files (a file containing information configured using Code Generator and a file containing information regarding the source code).
----------------------------	---






- Remarks 1.** The output format for the report file (either HTML or CSV) is selected by clicking [\[Generation\] tab >> \[Report type\]](#) in the [Property panel](#).
2. The destination folder for the report file is specified by clicking [\[Generation\] tab >> \[Output folder\]](#) in the [Property panel](#).

[[Help] menu (Code Generator Preview panel-dedicated items)]

Open Help for Code Generator Preview Panel	Displays the help of this panel.
--	----------------------------------

[Context menu]

The following context menu items are displayed by right clicking the mouse.

Generate code	<p>Makes a setting so that the source code of the currently selected API function is generated to the folder specified by selecting [Generation] tab >> [Output folder] in the Property panel.</p> <p>Selecting this context menu item changes the icon of the currently selected API function from  to .</p> <p>This item will be grayed out (disabled) if the currently selected API function is not initialization API function, and "Output only initialization API function" is selected [Generation] tab >> [Output control of API function] in the Property panel.</p>
Not generate code	<p>Makes a setting so that the source code of the currently selected API function is not generated when the  button is pressed in the Code Generator panel.</p> <p>Selecting this context menu item changes the icon of the currently selected API function from  to .</p>
Rename	<p>Changes the name portion of the currently selected file or API function into an edit box for editing the name.</p> <p>You can change the name of the file or API function by editing its name in the edit box.</p>
Default	Reverts the file name or API function name to its original name before it was edited.
Property	Opens the Property panel that contains the information for the currently selected file.

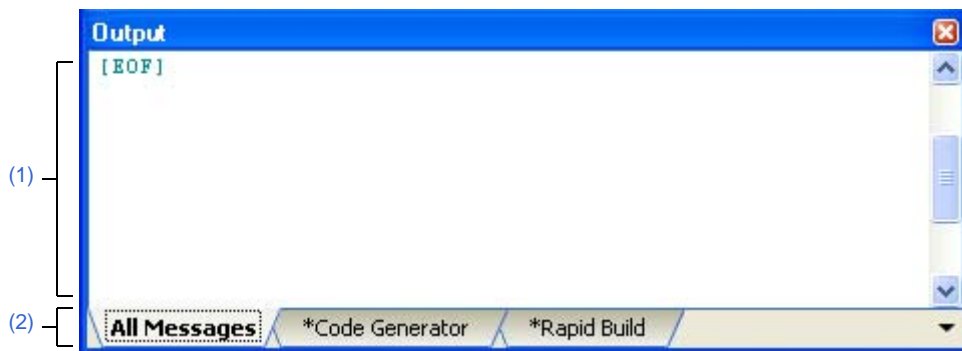
Output panel

This panel is used to display operation logs for various components (design tool, build tool, debug tool, etc.) provided by CubeSuite+.

The messages are classified by the message origination tool and displayed on the individual tabs.

Remark The Message area can be zoomed in and out by in the tool bar, or by operating the mouse wheel while holding down the [Ctrl] key.

Figure A-18. Output Panel



The following items are explained here.

- [How to open]
- [Description of each area]
- [[File] menu (Output panel-dedicated items)]
- [[Edit] menu (Output panel-dedicated items)]
- [Context menu]

[How to open]

- From the [View] menu, select [Output].

[Description of each area]

(1) Message area

The output messages of each tool are displayed.

The colors of message display differ with the type of message as shown below (character colors and background colors depend on the configuration in the [General - Font and Color] category of the Option dialog box).

Message Type	Display Example (Default)		Description	
Normal message		Character color	Black	Displayed with information notices.
		Background color	White	
Warning message		Character color	Bule	Displayed with warnings about operations.
		Background color	Standard color	
Error message		Character color	Red	Displayed when there is a critical error, or when execution is not possible due to a operational mistake.
		Background color	Light gray	

(2) Tab selection area

Select the tab that indicates the origin of message.

The following tabs are available for the debug tool.

Tab Name	Description
All Messages	Displays operation logs for all components (design tool, build tool, debug tool, etc.) provided by CubeSuite+ in order of output.
Code Generator	Display only operation logs for the Code Generator out of those for various components (design tool, build tool, debug tool, etc.) provided by CubeSuite+.

Caution Even if a new message is output on a deselected tab, tab selection will not automatically switch. In this case, " * " mark will be added in front of the tab name, indicating that a new message has been output.

[[File] menu (Output panel-dedicated items)]

Save Output- <i>Tab Name</i>	Saves the message corresponding to the specified tab overwriting the existing file.
Save Output- <i>Tab Name</i> As...	Opens the Save As dialog box for naming and saving the message corresponding to the specified tab.

[[Edit] menu (Output panel-dedicated items)]

Copy	Sends the character string or lines selected with range selection to the clipboard.
Select All	Selects all the messages displayed on the Message area .
Search...	Opens the Search and Replace dialog box for searching strings with the [Quick Search] tab selected.
Replace...	Opens the Search and Replace dialog box for replacing strings with the [Whole Replace] tab selected.

[Context menu]

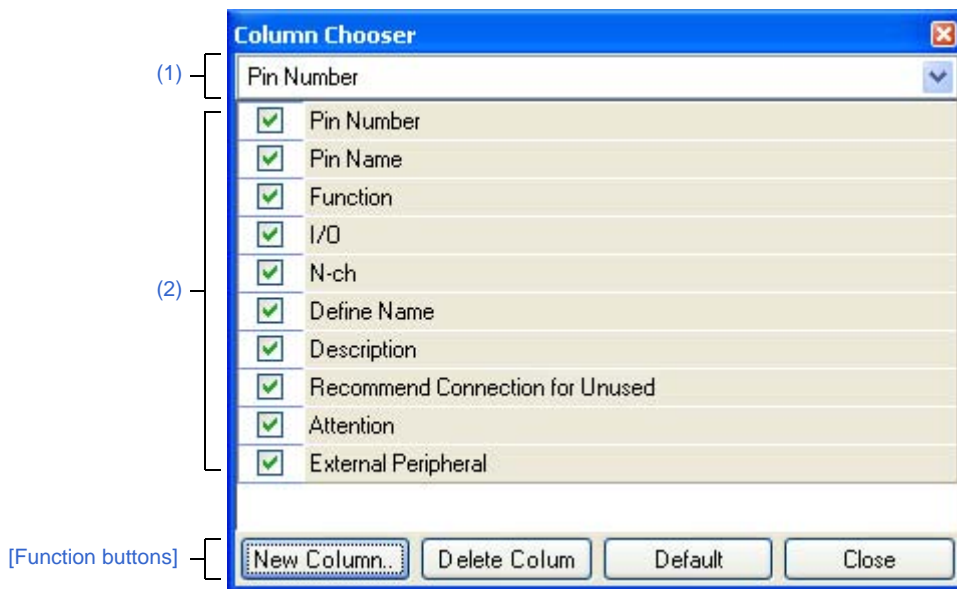
The following context menu items are displayed by right clicking the mouse.

Copy	Sends the character string or lines selected with range selection to the clipboard.
Select All	Selects all the messages displayed on the Message area .
Clear	Deletes all the messages displayed on the Message area .
Stop Searching	<p>Cancels the search currently being executed.</p> <p>This is invalid when a search is not being executed.</p>
Open Help for Message	<p>Displays help for the message on the current caret location.</p> <p>This only applies to warning messages and error messages.</p>

Column Chooser dialog box

This dialog box allows you to choose whether or not to display the item listed in this dialog box in the device pin list, and add columns to or delete columns from the device pin list.




Figure A-19. Column Chooser Dialog Box



The following items are explained here.

- [How to open]
- [Description of each area]
- [Function buttons]

[How to open]

- In the [Pin Number] tab of the Device Pin List panel, click the  button.
- In the [Macro] tab of the Device Pin List panel, click the  button.
- In the [External Peripheral] tab of the Device Pin List panel, click the  button.

[Description of each area]

(1) Operational object selection area

This area allows you to select the device pin list to be configured in this dialog box.

Pin Number	Configures the device pin list corresponding to the [Pin Number] tab.
Macro	Configures the device pin list belonging to the first layer of the [Macro] tab.
Macro - Pin	Configures the device pin list belonging to the second layer of the [Macro] tab.
External Peripheral	Configures the device pin list belonging to the first layer of the [External Peripheral] tab.
External Peripheral - Pin	Configures the device pin list belonging to the second layer of the [External Peripheral] tab.

Figure A-20. Operational Object ([Pin Number] Tab)

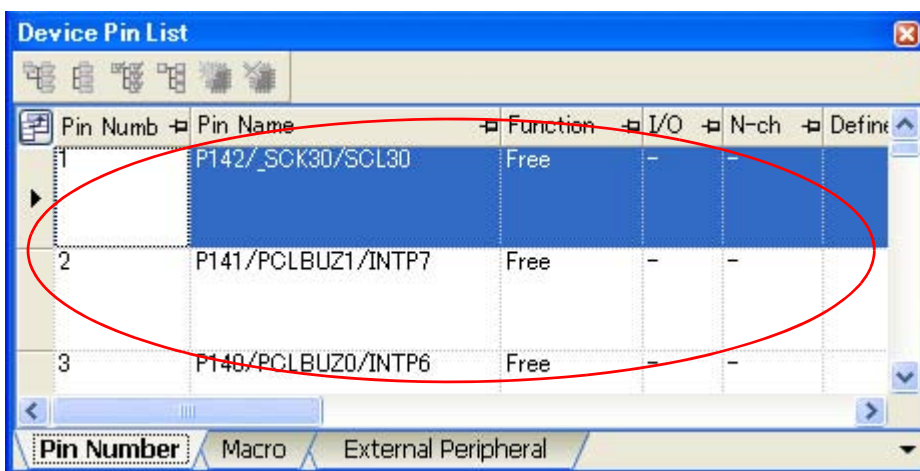


Figure A-21. Operational Object ([Macro] Tab: First Layer)

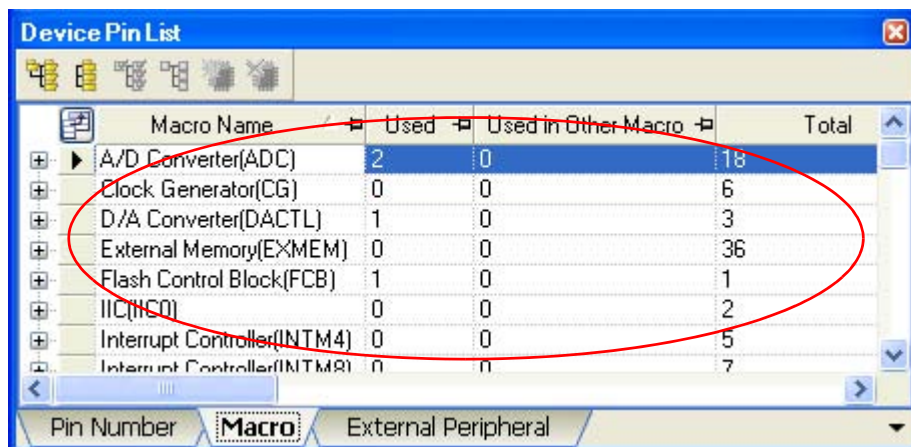


Figure A-22. Operational Object ([Macro] Tab: Second Layer)

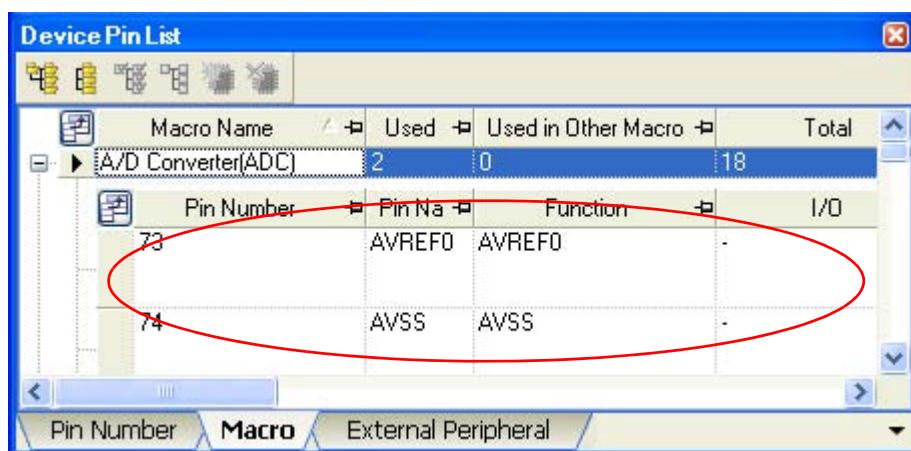


Figure A-23. Operational Object ([External Peripheral] Tab: First Layer)

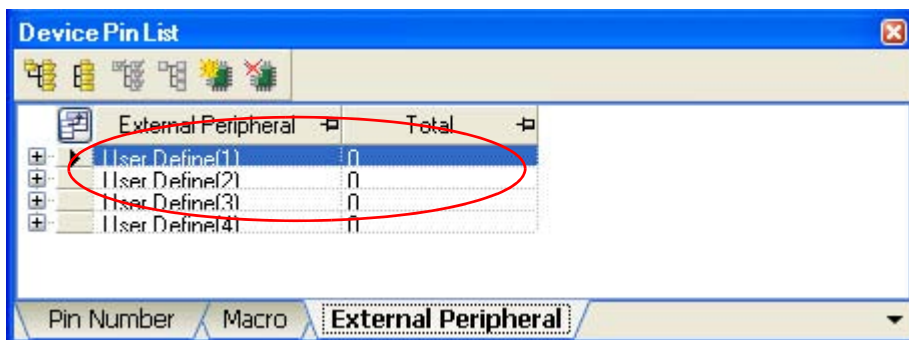
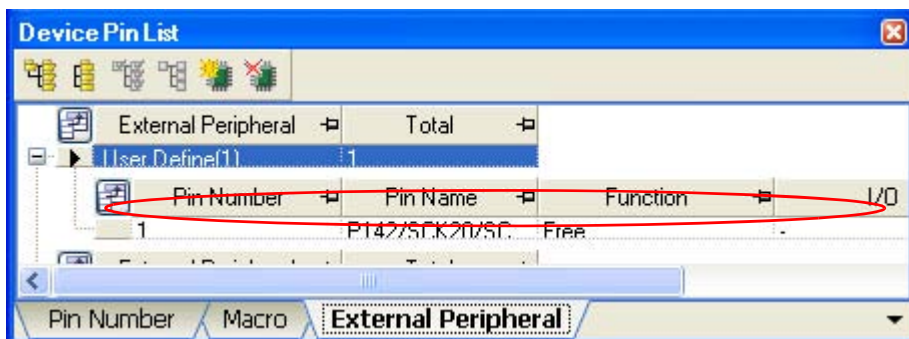


Figure A-24. Operational Object ([External Peripheral] Tab: Second Layer)



(2) Displayed item selection area

Select whether or not to display the item selected in the [Operational object selection area](#) in the device pin list.

Checked	Displays the selected item in the device pin list.
Not checked	Hides the selected item in the device pin list.

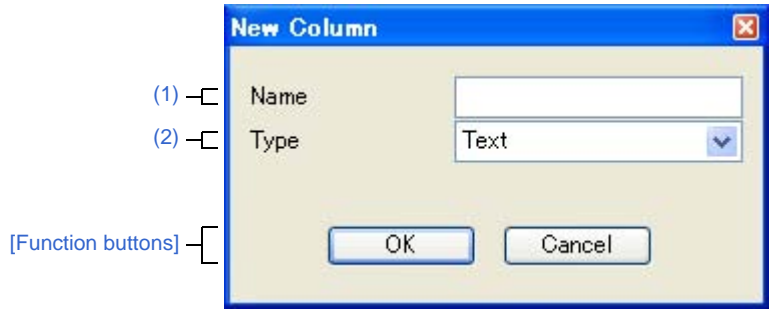
[Function buttons]

Button	Function
New Column...	Opens the New Column dialog box for adding columns to the device pin list.
Delete Column	Deletes the selected columns from the device pin list. You can only delete the column which you added using the New Column dialog box .
Default	Restores the column order to the default settings.
Close	Closes this dialog box.

New Column dialog box

This dialog box allows you to add your own column to the device pin list.

Figure A-25. New Column Dialog Box



The following items are explained here.

- [\[How to open\]](#)
- [\[Description of each area\]](#)
- [\[Function buttons\]](#)

[How to open]

- Click the [\[New Column...\]](#) button in the [Column Chooser dialog box](#).

[Description of each area]

(1) [Name]

This area allows you to enter column headings of the columns added to the device pin list. Within 256 characters can be entered in the [\[Name\]](#).

(2) [Type]

Select the input format of the column to add to the device pin list.

Text	Only character strings can be entered in the column.
Check box	Adds a column of check boxes.
Whole number	Only integers can be entered in the column.
Real number	Only real numbers can be entered in the column.
Date	Only dates in YYYYMMDD format can be entered in the column.

[Function buttons]

Button	Function
OK	Adds a column that has the column heading specified in the [Name] to the right end of the device pin list.
Cancel	Ignores the setting and closes this dialog box.

Browse For Folder dialog box

This dialog box allows you to specify the output destination for files (source code, report file, etc.).

Figure A-26. Browse For Folder Dialog Box



The following items are explained here.

- [How to open]
- [Description of each area]
- [Function buttons]

[How to open]

- In the [Generation] tab of the Property panel, click the [...] button in [Output folder].

[Description of each area]

(1) Folder location

Select the folder to which the files (source code, report file, etc.) are output.

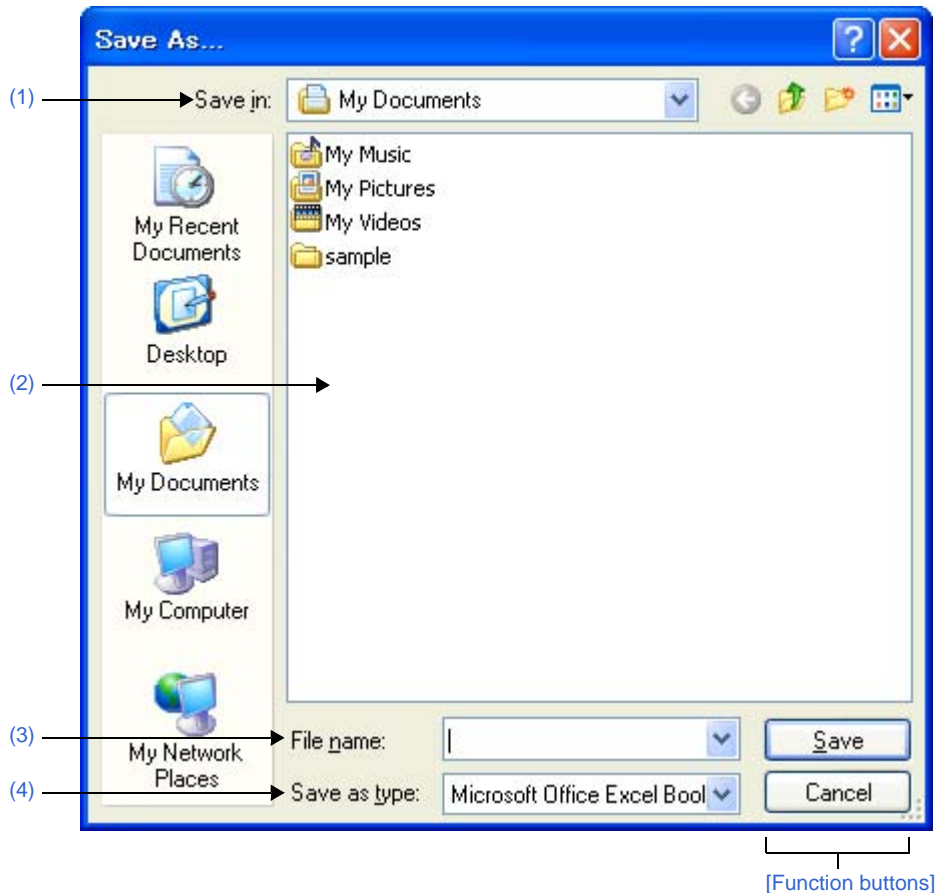
[Function buttons]

Button	Function
Make New Folder	Creates a "New Folder" below the folder selected in the Folder location.
OK	Specifies the folder selected in the Folder location as the destination for the files.
Cancel	Ignores the setting and closes this dialog box.

Save As dialog box

This dialog box allows you to name and save a file (such as a report file).

Figure A-27. Save As Dialog Box



The following items are explained here.

- [How to open]
- [Description of each area]
- [Function buttons]

[How to open]

- From the [File] menu, select [Save <object> As...].

[Description of each area]

(1) [Save in]

Select the folder to which the files (report files, etc.) are output.

(2) List of files

This area displays a list of files matching the conditions selected in [Save in] and [Save as type].

(3) [File name]

Specify the name of the file to be output.

(4) [Save as type]

Select the type of the file to be output.

Microsoft Office Excel Book (*.xls)	Microsoft Office Excel Book format
Bitmap (*.bmp)	Bitmap format
PNG (*.png)	PNG format
JPEG (*.jpg)	JPEG format
EMF (*.emf)	EMF format

[Function buttons]

Button	Function
Save	Outputs a file having the name specified in the [File name] and [Save as type] to the folder specified in the [Save in] .
Cancel	Ignores the setting and closes this dialog box.

APPENDIX B OUTPUT FILES

This appendix describes the files output by the Code Generator.

B.1 Overview

Below is a list of files output by the Code Generator.

Table B-1. File List

Unit of Output	File Name	Description
Peripheral function	r_cg_PeripheralFunctionName.c	Initial function, API function
	r_cg_PeripheralFunctionName_user.c	Interrupt function, callback function
	r_cg_PeripheralFunctionName.h	Defines macros for assigning values to registers
Project	r_main.c	main function, R_MAIN_UserInit function
	r_systeminit.c	Call initial function of peripheral function Call R_CGC_Get_ResetSource
	r_cg_macrodriver.h	Defines common macros used by all source files
	r_cg_userdefine.h	Empty file (for user definitions)
	r_lk.dr	Link directive

B.2 Output File

Below is a list of files (peripheral function) output by the Code Generator.

Table B-2. File List (Peripheral Function)

Peripheral Function	Source File Name	Names of API Functions Included
Clock Generator	r_cg_cgc.c	R_CGC_Create R_CGC_Set_ClockMode R_CGC_Set_CRCON
	r_cg_cgc_user.c	R_CGC_Create_UserInit r_cg_clockmonitor_interrupt R_CGC_Get_ResetSource
	r_cg_cgc.h	-
Port	r_cg_port.c	R_PORT_Create
	r_cg_port_user.c	R_PORT_Create_UserInit
	r_cg_port.h	-
Interrupt	r_cg_intc.c	R_INTC_Create R_INTCn_Start R_INTCn_Stop R_KEY_Create R_KEY_Start R_KEY_Stop R_INTCLRn_Start R_INTCLRn_Stop
	r_cg_intc_user.c	R_INTC_Create_UserInit

Peripheral Function	Source File Name	Names of API Functions Included
Interrupt	r_cg_intc_user.c	r_intcn_interrupt R_KEY_Create_UserInit r_key_interrupt r_intclrn_interrupt
	r_cg_intc.h	-
Serial	r_cg_serial.c	R_SAUm_Create R_SAUm_Set_PowerOff R_SAUm_Set_SnoozeOn R_SAUm_Set_SnoozeOff R_UARTn_Create R_UARTn_Start R_UARTn_Stop R_UARTn_Send R_UARTn_Receive R_CSImn_Create R_CSImn_Start R_CSImn_Stop R_CSImn_Send R_CSImn_Receive R_CSImn_Send_Receive R_IICmn_Create R_IICmn_StartCondition R_IICmn_StopCondition R_IICmn_Stop R_IICmn_Master_Send R_IICmn_Master_Receive R_DALIn_Create R_DALIn_Start R_DALIn_Stop R_DALIn_Send R_DALIn_Receive R_UARTFn_Create R_UARTFn_Start R_UARTFn_Stop R_UARTFn_Set_PowerOff R_UARTFn_Send R_UARTFn_Receive R_UARTFn_Set_DataComparisonOn R_UARTFn_Set_DataComparisonOff R_IICAn_Create R_IICAn_StopCondition R_IICAn_Stop R_IICAn_Set_PowerOff R_IICAn_Master_Send R_IICAn_Master_Receive R_IICAn_Slave_Send R_IICAn_Slave_Receive

Peripheral Function	Source File Name	Names of API Functions Included
Serial	r_cg_serial_user.c	R_SAUm_Create_UserInit r_uartn_interrupt_send r_uartn_interrupt_receive r_uartn_interrupt_error r_uartn_callback_sendend r_uartn_callback_receiveend r_uartn_callback_error r_uartn_callback_softwareoverrun r_csimn_interrupt r_csimn_callback_sendend r_csimn_callback_receiveend r_csimn_callback_error r_iicmn_interrupt r_iicmn_callback_master_sendend r_iicmn_callback_master_receiveend r_iicmn_callback_master_error r_dalin_interrupt_send r_dalin_interrupt_receive r_dalin_interrupt_error r_dalin_callback_sendend r_dalin_callback_receiveend r_dalin_callback_error r_dalin_callback_softwareoverrun R_UARTFn_Create_UserInit r_uartfn_interrupt_send r_uartfn_interrupt_receive r_uartfn_interrupt_error r_uartfn_callback_sendend r_uartfn_callback_receiveend r_uartfn_callback_error r_uartfn_callback_softwareoverrun r_uartfn_callback_expbtdetect r_uartfn_callback_idmatch R_IICAn_Create_UserInit r_iican_interrupt r_iican_callback_master_sendend r_iican_callback_master_receiveend r_iican_callback_master_error r_iican_callback_slave_sendend r_iican_callback_slave_receiveend r_iican_callback_slave_error r_iican_callback_getstopcondition
	r_cg_serial.h	-
A/D Converter	r_cg_adc.c	R_ADC_Create R_ADC_Set_OperationOn R_ADC_Set_OperationOff R_ADC_Start

Peripheral Function	Source File Name	Names of API Functions Included
A/D Converter	r_cg_adc.c	R_ADC_Stop R_ADC_Set_PowerOff R_ADC_Set_ADChannel R_ADC_Set_SnoozeOn R_ADC_Set_SnoozeOff R_ADC_Set_TestChannel R_ADC_Get_Result R_ADC_Get_Result_8bit
	r_cg_adc_user.c	R_ADC_Create_UserInit r_adc_interrupt
	r_cg_adc.h	-
D/A Converter	r_cg_dac.c	R_DAC_Create R_DACn_Start R_DACn_Stop R_DAC_Set_PowerOff R_DACn_Set_ConversionValue
	r_cg_dac_user.c	R_DAC_Create_UserInit
	r_cg_dac.h	-
Timer	r_cg_timer.c	R_TAUm_Create R_TAUm_Channeln_Start R_TAUm_Channeln_Higher8bits_Start R_TAUm_Channeln_Lower8bits_Start R_TAUm_Channeln_Stop R_TAUm_Channeln_Higher8bits_Stop R_TAUm_Channeln_Lower8bits_Stop R_TAUm_Set_PowerOff R_TAUm_Channeln_Get_PulseWidth R_TAUm_Channeln_Set_SoftwareTriggerOn R_TMR_RDn_Create R_TMR_RDn_Start R_TMR_RDn_Stop R_TMR_RDn_Set_PowerOff R_TMR_RDn_ForcedOutput_Start R_TMR_RDn_ForcedOutput_Stop R_TMR_RDn_Get_PulseWidth R_TMR_RG0_Create R_TMR_RG0_Start R_TMR_RG0_Stop R_TMR_RG0_Set_PowerOff R_TMR_RG0_Get_PulseWidth R_TMR_RJ0_Create R_TMR_RJ0_Start R_TMR_RJ0_Stop R_TMR_RJ0_Set_PowerOff R_TMR_RJ0_Get_PulseWidth R_TMR_KB_Create

Peripheral Function	Source File Name	Names of API Functions Included
Timer	r_cg_timer.c	R_TMR_KBm_Start R_TMR_KBm_Stop R_TMR_KBm_Set_PowerOff R_TMR_KBm_ForcedOutput_Start R_TMR_KBm_ForcedOutput_Stop R_TMR_KC0_Create R_TMR_KC0_Start R_TMR_KC0_Stop R_TMR_KC0_Set_PowerOff R_WUTM_Create R_WUTM_Start R_WUTM_Stop R_WUTM_Set_PowerOff
	r_cg_timer_user.c	R_TAUm_Create_UserInit r_taum_channeln_interrupt r_taum_channeln_higher8bits_interrupt R_TMR_RDn_Create_UserInit r_tmr_rdn_interrupt R_TMR_RG0_Create_UserInit r_tmr_rg0_interrupt R_TMR_RJ0_Create_UserInit r_tmr_rj0_interrupt R_TMR_KBm_Create_UserInit r_tmr_kbm_interrupt R_TMR_KC0_Create_UserInit r_tmr_kc0_interrupt R_WUTM_Create_UserInit r_wuttm_interrupt
	r_cg_timer.h	-
Watchdog Timer	r_cg_wdt.c	R_WDT_Create R_WDT_Restart
	r_cg_wdt_user.c	R_WDT_Create_UserInit r_wdt_interrupt
	r_cg_wdt.h	-
Real-time Clock	r_cg_rtc.c	R_RTC_Create R_RTC_Start R_RTC_Stop R_RTC_Set_PowerOff R_RTC_Set_HourSystem R_RTC_Set_CounterValue R_RTC_Get_CounterValue R_RTC_Set_ConstPeriodInterruptOn R_RTC_Set_ConstPeriodInterruptOff R_RTC_Set_AlarmOn R_RTC_Set_AlarmOff R_RTC_Set_AlarmValue

Peripheral Function	Source File Name	Names of API Functions Included
Real-time Clock	r_cg_rtc.c	R_RTC_Get_AlarmValue R_RTC_Set_RTC1HZOn R_RTC_Set_RTC1HZOff
	r_cg_rtc_user.c	R_RTC_Create_UserInit r_rtc_interrupt r_rtc_callback_constperiod r_rtc_callback_alarm
	r_cg_rtc.h	-
Interval Timer	r_cg_it.c	R_IT_Create R_IT_Start R_IT_Stop R_IT_Set_PowerOff
	r_cg_it_user.c	R_IT_Create_UserInit r_it_interrupt
	r_cg_it.h	-
Comparator	r_cg_comp.c	R_COMP_Create R_COMPn_Start R_COMPn_Stop
	r_cg_comp_user.c	R_COMP_Create_UserInit r_compn_interrupt
	r_cg_comp.h	-
Clock Output/Buzzer Output	r_cg_pclbuz.c	R_PCLBUZn_Create R_PCLBUZn_Start R_PCLBUZn_Stop R_PCLBUZ_Set_PowerOff
	r_cg_pclbuz_user.c	R_PCLBUZn_Create_UserInit
	r_cg_pclbuz.h	-
Data Transfer Controller	r_cg_dtc.c	R_DTC_Create R_DTCn_Start R_DTCn_Stop R_DTC_Set_PowerOff
	r_cg_dtc_user.c	R_DTC_Create_UserInit
	r_cg_dtc.h	-
Event Link Controller	r_cg_elc.c	R_ELC_Create R_ELC_Stop
	r_cg_elc_user.c	R_ELC_Create_UserInit
	r_cg_elc.h	-
LCD Controller/Driver	r_cg_lcd.c	R_LCD_Create R_LCD_Start R_LCD_Stop R_LCD_Set_VoltageOn R_LCD_Set_VoltageOff R_LCD_Set_PowerOff

Peripheral Function	Source File Name	Names of API Functions Included
LCD Controller/Driver	r_cg_lcd_user.c	R_LCD_Create_UserInit r_lcd_interrupt
	r_cg_lcd.h	-
Sound Generator	r_cg_sg.c	R_SG_Create R_SG_Start R_SG_Stop
	r_cg_sg_user.c	R_SG_Create_UserInit r_sg_interrupt
	r_cg_sg.h	-
DMA Controller	r_cg_dmac.c	R_DMACn_Create R_DMAC_Create R_DMACn_Start R_DMACn_Stop R_DMACn_Set_SoftwareTriggerOn
	r_cg_dmac_user.c	R_DMACn_Create_UserInit R_DMAC_Create_UserInit r_dmacn_interrupt
	r_cg_dmac.h	-
Voltage Detector	r_cg_lvd.c	R_LVD_Create R_LVD_InterruptMode_Start
	r_cg_lvd_user.c	R_LVD_Create_UserInit r_lvd_interrupt
	r_cg_lvd.h	-
Programmable Gain Amplifier	r_cg_pga.c	R_PGA_Create R_PGA_Start R_PGA_Stop
	r_cg_pga_user.c	R_PGA_Create_UserInit
	r_cg_pga.h	-

APPENDIX C API FUNCTIONS

This appendix describes the API functions output by the Code Generator.

C.1 Overview

Below are the naming conventions for API functions output by the Code Generator.

- Macro names are in ALL CAPS.
The number in front of the macro name is a hexadecimal value; this is the same value as the macro value.
- Local variable names are in all lower case.
- Global variable names start with a "g" and use Camel Case.
- Names of pointers to global variables start with a "gp" and use Camel Case.
- Names of elements in enum statements are in ALL CAPS.

C.2 Output Function

Below is a list of API functions output by the Code Generator.

Table C-1. API Function List

Peripheral Function	API Function Name	Function
Clock Generator	R_CGC_Create	Performs initialization required to control the clock generator, on-chip debug and etc..
	R_CGC_Create_UserInit	Performs user-defined initialization relating to the clock generator, on-chip debug, and etc..
	r_cgc_clockmonitor_interrupt	Performs processing in response to the clock monitor interrupt INTCLM.
	R_CGC_Get_ResetSource	Performs processing in response to RESET signal.
	R_CGC_Set_ClockMode	Changes the CPU clock/peripheral hardware clock.
	R_CGC_Set_CRCon	Starts the CRC operation function.
Port	R_PORT_Create	Performs initialization necessary to control port functions.
	R_PORT_Create_UserInit	Performs user-defined initialization relating to the port.
Interrupt	R_INTC_Create	Performs initialization necessary to control the external maskable interrupts INTP _n (and the external maskable interrupts INTPLR _n) functions.
	R_INTC_Create_UserInit	Performs user-defined initialization relating to the external maskable interrupts INTP _n (and the external maskable interrupts INTPLR _n) functions.
	r_intcn_interrupt	Performs processing in response to the external maskable interrupt INTP _n .
	R_INTCn_Start	Enables the acceptance of the external maskable interrupts INTP _n .
	R_INTCn_Stop	Disables the acceptance of the external maskable interrupts INTP _n .
	R_KEY_Create	Performs initialization necessary to control the key interrupt INTKR functions.
	R_KEY_Create_UserInit	Performs user-defined initialization relating to the key interrupt INTKR functions.

Peripheral Function	API Function Name	Function
Interrupt	r_key_interrupt	Performs processing in response to the key interrupt INTKR.
	R_KEY_Start	Enables the acceptance of the key interrupt INTKR.
	R_KEY_Stop	Disables the acceptance of the key interrupts INTKR.
	r_intclm_interrupt	Performs processing in response to the external maskable interrupt INTPLR n .
	R_INTCLRn_Start	Enables the acceptance of the external maskable interrupts INTPLR n .
	R_INTCLRn_Stop	Disables the acceptance of the external maskable interrupts INTPLR n .
Serial	R_SAUm_Create	Performs initialization necessary to control the serial array unit and serial interface functions.
	R_SAUm_Create_UserInit	Performs user-defined initialization related to the serial array unit and serial interface functions.
	R_SAUm_Set_PowerOff	Halts the clock supplied to the serial array unit.
	R_SAUm_Set_SnoozeOn	Enables the switch from STOP mode to SNOOZE mode.
	R_SAUm_Set_SnoozeOff	Disables the switch from STOP mode to SNOOZE mode.
	R_UARTn_Create	Performs initialization of the serial interface (UART) channel.
	r_uartn_interrupt_send	Performs processing in response to the UART transmission end interrupt INTST n .
	r_uartn_interrupt_receive	Performs processing in response to the UART reception end interrupt INTSR n .
	r_uartn_interrupt_error	Performs processing in response to the reception error interrupt INTSRE n .
	R_UARTn_Start	Sets UART communication to standby mode.
	R_UARTn_Stop	Ends UART communication.
	R_UARTn_Send	Starts UART data transmission.
	R_UARTn_Receive	Starts UART data reception.
	r_uartn_callback_sendend	Performs processing in response to the UART transmission end interrupt INTST n .
	r_uartn_callback_receiveend	Performs processing in response to the UART reception end interrupt INTSR n .
	r_uartn_callback_error	Performs processing in response to the UART reception error interrupt INTSRE n .
	r_uartn_callback_softwareoverrun	Performs processing in response to detection of overrun error.
	R_CSImn_Create	Performs initialization of the serial interface (CSI) channel.
	r_csimn_interrupt	Performs processing in response to the CSI communication end interrupt INTCSIm n .
	R_CSImn_Start	Sets CSI communication to standby mode.
	R_CSImn_Stop	Ends CSI communication.
	R_CSImn_Send	Starts CSI data transmission.
	R_CSImn_Receive	Starts CSI data reception.

Peripheral Function	API Function Name	Function
Serial	R_CSImn_Send_Receive	Starts CSI data transmission/reception.
	r_csimn_callback_sendend	Performs processing in response to the CSI transmission end interrupt INTCSImn.
	r_csimn_callback_receiveend	Performs processing in response to the CSI reception end interrupt INTCSImn.
	r_csimn_callback_error	Performs processing in response to the CSI reception error interrupt INTSREn.
	R_IICmn_Create	Performs initialization of the serial interface (simple IIC) channel.
	r_iicmn_interrupt	Performs processing in response to the simple IIC communication end interrupt INTIICmn.
	R_IICmn_StartCondition	Generates start conditions.
	R_IICmn_StopCondition	Generates stop conditions.
	R_IICmn_Stop	Ends simple IIC communication.
	R_IICmn_Master_Send	Starts simple IIC master transmission.
	R_IICmn_Master_Receive	Starts simple IIC master reception.
	r_iicmn_callback_master_sendend	Performs processing in response to the simple IICmn master transmission end interrupt INTIICmn.
	r_iicmn_callback_master_receiveend	Performs processing in response to the simple IICmn master reception end interrupt INTIICmn.
	r_iicmn_callback_master_error	Performs processing in response to detection of parity error (ACK error).
	R_DALIn_Create	Performs initialization of the serial interface (DALI) channel.
	r_dalin_interrupt_send	Performs processing in response to the DALI transmission end interrupt INTSTDLn.
	r_dalin_interrupt_receive	Performs processing in response to the DALI reception end interrupt INTSRDLn.
	r_dalin_interrupt_error	Performs processing in response to the DALI reception error interrupt INTSREDLn.
	R_DALIn_Start	Sets DALI communication to standby mode.
	R_DALIn_Stop	Ends DALI communication.
	R_DALIn_Send	Starts DALI data transmission.
	R_DALIn_Receive	Starts DALI data reception.
	r_dalin_callback_sendend	Performs processing in response to the DALI transmission end interrupt INTSTDLn.
	r_dalin_callback_receiveend	Performs processing in response to the DALI reception end interrupt INTSRDLn.
	r_dalin_callback_error	Performs processing in response to the DALI reception error interrupt INTSREDLn.
	r_dalin_callback_softwareoverrun	Performs processing in response to detection of overrun error.
	R_UARTFn_Create	Performs initialization of the asynchronous serial interface (UARTF) channel.

Peripheral Function	API Function Name	Function
Serial	R_UARTFn_Create_UserInit	Performs user-defined initialization of the asynchronous serial interface (UARTF) channel.
	r_uartfn_interrupt_send	Performs processing in response to the LIN-UART transmission end interrupt INTLT.
	r_uartfn_interrupt_receive	Performs processing in response to the LIN-UART reception end interrupt INTLR.
	r_uartfn_interrupt_error	Performs processing in response to the LIN-UART reception status interrupt INTLS.
	R_UARTFn_Start	Sets UARTF communication to standby mode.
	R_UARTFn_Stop	Ends UARTF communication.
	R_UARTFn_Set_PowerOff	Halts the clock supplied to the asynchronous serial interface (UARTF) channel.
	R_UARTFn_Send	Starts UARTF data transmission.
	R_UARTFn_Receive	Starts UARTF data reception.
	R_UARTFn_Set_DataComparison On	Starts the data comparison.
	R_UARTFn_Set_DataComparison Off	Ends the data comparison.
	r_uartfn_callback_sendend	Performs processing in response to the LIN-UART transmission end interrupt INTLT.
	r_uartfn_callback_receiveend	Performs processing in response to the LIN-UART reception end interrupt INTLR.
	r_uartfn_callback_error	Performs processing in response to the LIN-UART reception status interrupt INTLS.
	r_uartfn_callback_softwareoverrun	Performs processing in response to detection of overrun error.
	r_uartfn_callback_expbitdetect	Performs processing in response to detection of expansion bit.
	r_uartfn_callback_idmatch	Performs processing in response to match of ID parity.
	R_IICAn_Create	Performs initialization of the serial interface (IICA) channel.
	R_IICAn_Create_UserInit	Performs user-defined initialization of the serial interface (IICA) channel.
	r_iican_interrupt	Performs processing in response to the IICA communication end interrupt INTIICAn.
	R_IICAn_StopCondition	Generates stop conditions.
	R_IICAn_Stop	Ends IICA communication.
	R_IICAn_Set_PowerOff	Halts the clock supplied to the serial interface (IICA) channel.
	R_IICAn_Master_Send	Starts IICA master transmission.
	R_IICAn_Master_Receive	Starts IICA master reception.
	r_iican_callback_master_sendend	Performs processing in response to the IICA master transmission end interrupt INTIICAn.
	r_iican_callback_master_receiveend	Performs processing in response to the IICA master reception end interrupt INTIICAn.

Peripheral Function	API Function Name	Function
Serial	r_iican_callback_master_error	Performs processing in response to detection of IICA master communication error.
	R_IICAn_Slave_Send	Starts IICA slave transmission.
	R_IICAn_Slave_Receive	Starts IICA slave reception.
	r_iican_callback_slave_sendend	Performs processing in response to the IICA slave transmission end interrupt INTIICAn.
	r_iican_callback_slave_receiveend	Performs processing in response to the IICA slave reception end interrupt INTIICAn.
	r_iican_callback_slave_error	Performs processing in response to detection of IICA slave communication error.
	r_iican_callback_getstopcondition	Performs processing in response to detection of stop condition.
A/D Converter	R_ADC_Create	Performs initialization necessary to control A/D converter functions.
	R_ADC_Create_UserInit	Performs user-defined initialization relating to the A/D converter.
	r_adc_interrupt	Performs processing in response to the A/D conversion end interrupt INTAD.
	R_ADC_Set_OperationOn	Enables operation of voltage converter.
	R_ADC_Set_OperationOff	Disables operation of voltage converter.
	R_ADC_Start	Starts A/D conversion.
	R_ADC_Stop	Ends A/D conversion.
	R_ADC_Set_PowerOff	Halts the clock supplied to the A/D converter.
	R_ADC_Set_ADChannel	Configures the analog voltage input pin for A/D conversion.
	R_ADC_Set_SnoozeOn	Enables the switch from STOP mode to SNOOZE mode.
	R_ADC_Set_SnoozeOff	Disables the switch from STOP mode to SNOOZE mode.
	R_ADC_Set_TestChannel	Sets the operation mode of A/D converter.
	R_ADC_Get_Result	Reads the results of A/D conversion (10-bit).
	R_ADC_Get_Result_8bit	Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).
D/A Converter	R_DAC_Create	Performs initialization necessary to control the D/A converter functions.
	R_DAC_Create_UserInit	Performs user-defined initialization relating to the D/A converter.
	R_DACn_Start	Starts D/A conversion.
	R_DACn_Stop	Ends D/A conversion.
	R_DAC_Set_PowerOff	Halts the clock supplied to the D/A converter.
	R_DACn_Set_ConversionValue	Sets the analog voltage output to the ANOn pin.
Timer	R_TAUm_Create	Performs initialization necessary to control timer array unit functions.
	R_TAUm_Create_UserInit	Performs user-defined initialization relating to the timer array unit.

Peripheral Function	API Function Name	Function
Timer	r_taum_channeln_interrupt	Performs processing in response to the timer interrupt $INTTm_n$.
	r_taum_channeln_higher8bits_interrupt	Performs processing in response to the timer interrupt $INTTm_nH$.
	R_TAUm_Channeln_Start	Starts the count for channel n .
	R_TAUm_Channeln_Higher8bits_Start	Starts the count (higher 8-bit) for channel 1 or channel 3.
	R_TAUm_Channeln_Lower8bits_Start	Starts the count (lower 8-bit) for channel 1 or channel 3.
	R_TAUm_Channeln_Stop	Ends the count for channel n .
	R_TAUm_Channeln_Higher8bits_Stop	Ends the count (higher 8-bit) for channel 1 or channel 3.
	R_TAUm_Channeln_Lower8bits_Stop	Ends the count (lower 8-bit) for channel 1 or channel 3.
	R_TAUm_Set_PowerOff	Halts the clock supplied to the timer array unit.
	R_TAUm_Channeln_Get_PulseWidth	Captures the high/low-level width measured between pulses of the signal (pulses) input to the Tl_m_n pin.
	R_TAUm_Channeln_Set_SoftwareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.
	R_TMR_RDn_Create	Performs initialization necessary to control the 16-bit timer RD_n functions.
	R_TMR_RDn_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RD_n .
	r_tmr_rdn_interrupt	Performs processing in response to the timer interrupt.
	R_TMR_RDn_Start	Starts the count for 16-bit timer RD_n .
	R_TMR_RDn_Stop	Ends the count for 16-bit timer RD_n .
	R_TMR_RDn_Set_PowerOff	Halts the clock supplied to the 16-bit timer RD_n .
	R_TMR_RDn_ForcedOutput_Start	Starts the pulse output forced cutoff for 16-bit timer RD_n .
	R_TMR_RDn_ForcedOutput_Stop	Ends the pulse output forced cutoff for 16-bit timer RD_n .
	R_TMR_RDn_Get_PulseWidth	Reads the pulse width of the 16-bit timer RD_n .
	R_TMR_RG0_Create	Performs initialization necessary to control the 16-bit timer RG_0 functions.
	R_TMR_RG0_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RG_0 .
	r_tmr_rg0_interrupt	Performs processing in response to the timer interrupt.
	R_TMR_RG0_Start	Starts the count for 16-bit timer RG_0 .
	R_TMR_RG0_Stop	Ends the count for 16-bit timer RG_0 .
	R_TMR_RG0_Set_PowerOff	Halts the clock supplied to the 16-bit timer RG_0 .
	R_TMR_RG0_Get_PulseWidth	Reads the pulse width of the 16-bit timer RG_0 .
	R_TMR_RJ0_Create	Performs initialization necessary to control the 16-bit timer RJ_0 functions.

Peripheral Function	API Function Name	Function
Timer	R_TMR_RJ0_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RJ0.
	r_tmr_rj0_interrupt	Performs processing in response to the timer interrupt.
	R_TMR_RJ0_Start	Starts the count for 16-bit timer RJ0.
	R_TMR_RJ0_Stop	Ends the count for 16-bit timer RJ0.
	R_TMR_RJ0_Set_PowerOff	Halts the clock supplied to the 16-bit timer RJ0.
	R_TMR_RJ0_Get_PulseWidth	Reads the pulse width of the 16-bit timer RJ0.
	R_TMR_KBm_Create	Performs initialization necessary to control the 16-bit timer KBm functions.
	R_TMR_KBm_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer KBm.
	r_tmr_kbm_interrupt	Performs processing in response to the timer interrupt.
	R_TMR_KBm_Start	Starts the count for 16-bit timer KBm.
	R_TMR_KBm_Stop	Ends the count for 16-bit timer KBm.
	R_TMR_KBm_Set_PowerOff	Halts the clock supplied to the 16-bit timer KBm.
	R_TMR_KBm_ForcedOutput_Start	Enables input of the trigger signal used for the forced output stop function.
	R_TMR_KBm_ForcedOutput_Stop	Disables input of the trigger signal used for the forced output stop function.
	R_TMR_KC0_Create	Performs initialization necessary to control the 16-bit timer KC0 functions.
	R_TMR_KC0_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer KC0.
	r_tmr_kc0_interrupt	Performs processing in response to the timer interrupt.
	R_TMR_KC0_Start	Starts the count for 16-bit timer KC0.
	R_TMR_KC0_Stop	Ends the count for 16-bit timer KC0.
	R_TMR_KC0_Set_PowerOff	Halts the clock supplied to the 16-bit timer KC0.
	R_WUTM_Create	Performs initialization necessary to control the 16-bit wakeup timer WUTM functions.
	R_WUTM_Create_UserInit	Performs user-defined initialization relating to the 16-bit wakeup timer WUTM.
	r_wutm_interrupt	Performs processing in response to the timer interrupt.
	R_WUTM_Start	Starts the count for 16-bit wakeup timer WUTM.
R_WUTM_Stop	Ends the count for 16-bit wakeup timer WUTM.	
R_WUTM_Set_PowerOff	Halts the clock supplied to the 16-bit wakeup timer WUTM.	
Watchdog Timer	R_WDT_Create	Performs initialization necessary to control watchdog timer functions.
	R_WDT_Create_UserInit	Performs user-defined initialization relating to the watchdog timer.
	r_wdt_interrupt	Performs processing in response to the interval interrupt INTWDTI.

Peripheral Function	API Function Name	Function
Watchdog Timer	R_WDT_Restart	Clears the watchdog timer counter and resumes counting.
Real-time Clock	R_RTC_Create	Performs initialization necessary to control real-time clock functions.
	R_RTC_Create_UserInit	Performs user-defined initialization relating to the real-time clock.
	r_rtc_interrupt	Performs processing in response to the real-time clock interrupt INTRTC.
	R_RTC_Start	Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second).
	R_RTC_Stop	Ends the count of the real-time clock (year, month, weekday, day, hour, minute, second).
	R_RTC_Set_PowerOff	Halts the clock supplied to the real-time clock.
	R_RTC_Set_HourSystem	Sets the clock type (12-hour or 24-hour clock) of the real-time clock.
	R_RTC_Set_CounterValue	Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.
	R_RTC_Get_CounterValue	Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.
	R_RTC_Set_ConstPeriodInterruptOn	Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.
	R_RTC_Set_ConstPeriodInterruptOff	Ends the cyclic interrupt function.
	r_rtc_callback_constperiod	Performs processing in response to the cyclic interrupt INTRTC.
	R_RTC_Set_AlarmOn	Starts the alarm interrupt function.
	R_RTC_Set_AlarmOff	Ends the alarm interrupt function.
	R_RTC_Set_AlarmValue	Sets the alarm conditions (weekday, hour, minute).
	R_RTC_Get_AlarmValue	Reads the alarm conditions (weekday, hour, minute).
	r_rtc_callback_alarm	Performs processing in response to the alarm interrupt INTRTC.
	R_RTC_Set_RTC1HZOn	Enables output of the correction clock (1 Hz) to the RTC1HZ pin.
R_RTC_Set_RTC1HZOff	Disables output of the correction clock (1 Hz) to the RTC1HZ pin.	
Interval Timer	R_IT_Create	Performs initialization necessary to control interval timer functions.
	R_IT_Create_UserInit	Performs user-defined initialization relating to the interval timer.
	r_it_interrupt	Performs processing in response to the interval timer interrupt INTIT.
	R_IT_Start	Starts the count of the interval timer.
	R_IT_Stop	Ends the count of the interval timer.
	R_IT_Set_PowerOff	Halts the clock supplied to the interval timer.

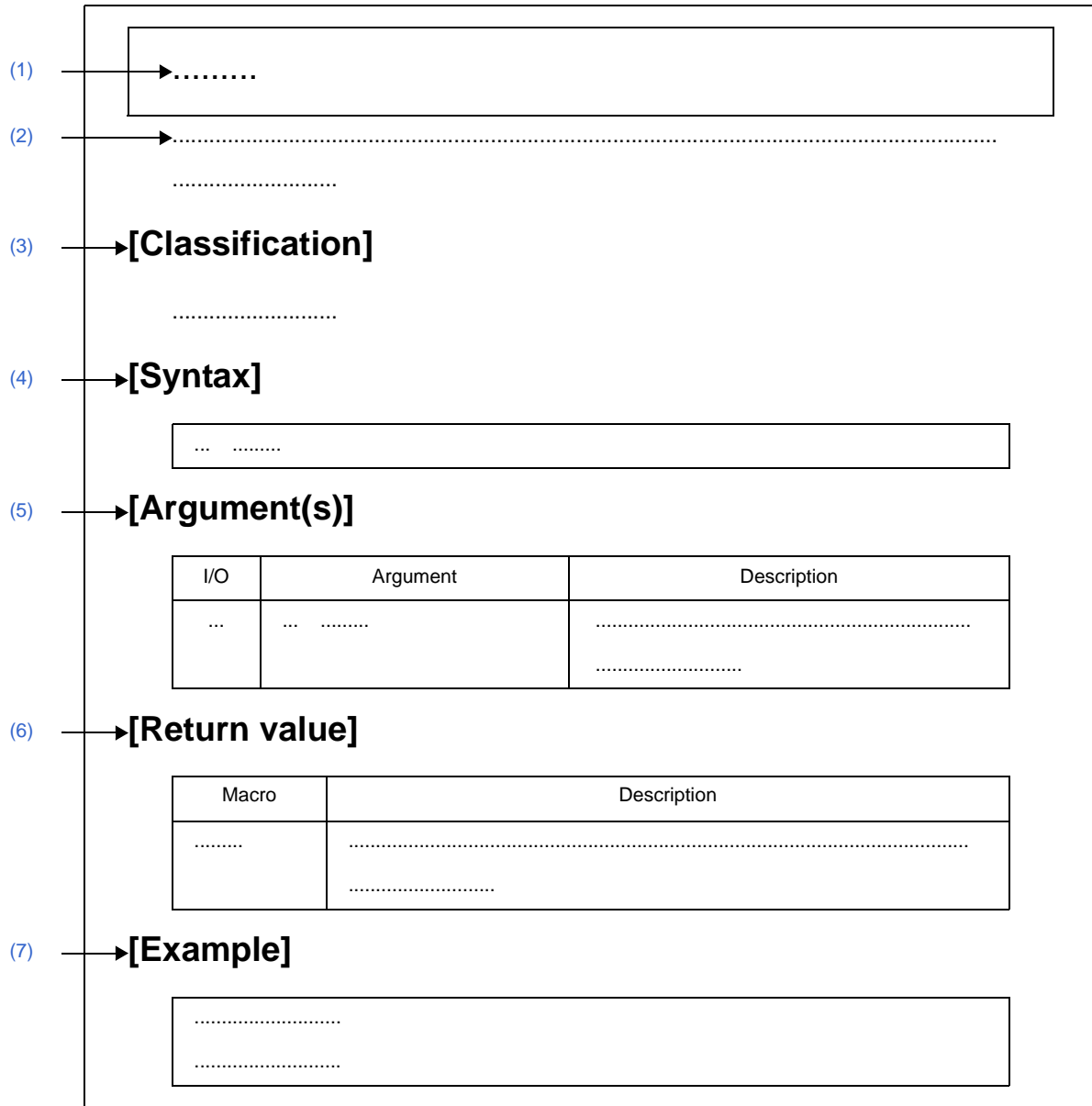
Peripheral Function	API Function Name	Function
Comparator	R_COMP_Create	Performs initialization necessary to control the comparator functions.
	R_COMP_Create_UserInit	Performs user-defined initialization relating to the comparator.
	r_compn_interrupt	Performs processing in response to the comparator interrupt INTCMPn.
	R_COMPn_Start	Begins comparison of reference input voltage and analog input voltage.
	R_COMPn_Stop	Stops comparison of reference input voltage and analog input voltage.
Clock Output/Buzzer Output	R_PCLBUZn_Create	Performs initialization necessary to control clock/buzzer output control circuit functions.
	R_PCLBUZn_Create_UserInit	Performs user-defined initialization relating to the clock/buzzer output control circuits.
	R_PCLBUZn_Start	Starts clock/buzzer output.
	R_PCLBUZn_Stop	Ends clock/buzzer output.
	R_PCLBUZ_Set_PowerOff	Halts the clock supplied to the clock/buzzer output controller.
Data Transfer Controller	R_DTC_Create	Performs initialization necessary to control the data transfer controller functions.
	R_DTC_Create_UserInit	Performs user-defined initialization relating to the data transfer controller.
	R_DTCn_Start	Enables operation of the data transfer controller.
	R_DTCn_Stop	Disables operation of the data transfer controller.
	R_DTC_Set_PowerOff	Halts the clock supplied to the data transfer controller.
Event Link Controller	R_ELC_Create	Performs initialization necessary to control the event link controller functions.
	R_ELC_Create_UserInit	Performs user-defined initialization relating to the event link controller.
	R_ELC_Stop	Disables operation of the event link controller.
LCD Controller/Driver	R_LCD_Create	Performs initialization necessary to control LCD controller/driver functions.
	R_LCD_Create_UserInit	Performs user-defined initialization relating to the LCD controller/driver.
	r_lcd_interrupt	Performs processing in response to the LCD frame interrupt INTLCD.
	R_LCD_Start	Sets the LCD controller/driver to display on status.
	R_LCD_Stop	Sets the LCD controller/driver to display off status.
	R_LCD_Set_VoltageOn	Enables operation of internal voltage boost circuit and capacitor split circuit.
	R_LCD_Set_VoltageOff	Disables operation of internal voltage boost circuit and capacitor split circuit.
	R_LCD_Set_PowerOff	Halts the clock supplied to the LCD controller/driver.
Sound Generator	R_SG_Create	Performs initialization necessary to control sound generator functions.

Peripheral Function	API Function Name	Function
Sound Generator	R_SG_Create_UserInit	Performs user-defined initialization relating to the sound generator.
	r_sg_interrupt	Performs processing in response to the threshold value detection of the logarithmic decrement interrupt INTSG.
	R_SG_Start	Enables operation of sound generator.
	R_SG_Stop	Disables operation of sound generator.
DMA Controller	R_DMACn_Create	Performs initialization necessary to control DMA controller functions.
	R_DMACn_Create_UserInit	Performs user-defined initialization relating to the DMA controller.
	R_DMAC_Create	Performs initialization necessary to control DMA controller functions.
	R_DMAC_Create_UserInit	Performs user-defined initialization relating to the DMA controller.
	r_dmacn_interrupt	This API function is called as the interrupt process corresponding to the interval interrupt INTWDTI.
	R_DMACn_Start	Enables operation of channel <i>n</i> .
	R_DMACn_Stop	Disables operation of channel <i>n</i> .
	R_DMACn_Set_SoftwareTriggerOn	Starts DMA transfer.
Voltage Detector	R_LVD_Create	Performs initialization necessary to control voltage detector functions.
	R_LVD_Create_UserInit	Performs user-defined initialization relating to the voltage detector.
	r_lvd_interrupt	Performs processing in response to the voltage detection interrupt INTLVI.
	R_LVD_InterruptMode_Start	Starts voltage detection (when in interrupt mode, and interrupt & reset mode).
Programmable Gain Amplifier	R_PGA_Create	Performs initialization necessary to control programmable gain amplifiers functions.
	R_PGA_Create_UserInit	Performs user-defined initialization relating to the programmable gain amplifiers.
	R_PGA_Start	Starts the operation of programmable gain amplifier.
	R_PGA_Stop	Ends the operation of programmable gain amplifier.

C.3 Function Reference

This section describes the API functions output by the Code Generator, using the following notation format.

Figure C-1. Notation Format of API Functions



(1) **Name**

Indicates the name of the API function.

(2) **Outline**

Outlines the functions of the API function.

(3) **[Classification]**

Indicates the name of the C source file to which the API function is output.

(4) **[Syntax]**

Indicates the format to be used when describing an API function to be called in C language.

(5) [Argument(s)]

API function arguments are explained in the following format.

I/O	Argument	Description
(a)	(b)	(c)

(a) I/O

Argument classification

I ... Input argument

O ... Output argument

(b) Argument

Argument data type

(c) Description

Description of argument

(6) [Return value]

API function return value is explained in the following format.

Macro	Description
(a)	(b)

(a) Macro

Macro of return value

(b) Description

Description of return value

(7) [Example]

Shows an example of the API function in use.

C.3.1 Clock Generator

Below is a list of API functions output by the Code Generator for clock generator use.

Table C-2. API Functions: [Clock Generator]

API Function Name	Function
R_CGC_Create	Performs initialization required to control the clock generator, on-chip debug and etc..
R_CGC_Create_UserInit	Performs user-defined initialization relating to the clock generator, on-chip debug, and etc..
r_cgc_clockmonitor_interrupt	Performs processing in response to the clock monitor interrupt INTCLM.
R_CGC_Get_ResetSource	Performs processing in response to RESET signal.
R_CGC_Set_ClockMode	Changes the CPU clock/peripheral hardware clock.
R_CGC_Set_CRCOn	Starts the CRC operation function.

R_CGC_Create

Performs initialization required to control the clock generator, on-chip debug and etc..

[Classification]

r_cg_cgc.c

[Syntax]

```
void R_CGC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_Create_UserInit

Performs user-defined initialization relating to the clock generator, on-chip debug, and etc..

Remark This API function is called as the [R_CGC_Create](#) callback routine.

[Classification]

r_cg_cgc_user.c

[Syntax]

```
void R_CGC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cgc_clockmonitor_interrupt

Performs processing in response to the clock monitor interrupt INTCLM.

Remark This API function is called as the interrupt process corresponding to the clock monitor interrupt INTCLM.

[Classification]

r_cg_cgc_user.c

[Syntax]

```
__interrupt static void r_cgc_clockmonitor_interrupt ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_CGC_Get_ResetSource

Performs processing in response to RESET signal.

[Classification]

r_cg_cgc_user.c

[Syntax]

```
void R_CGC_Get_ResetSource ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_Set_ClockMode

Changes the CPU clock/peripheral hardware clock.

[Classification]

r_cg_cgc.c

[Syntax]

```
#include "r_cg_macrodriver.h"
#include "r_cg_cgc.h"
MD_STATUS R_CGC_Set_ClockMode ( clock_mode_t mode );
```

[Argument(s)]

I/O	Argument	Description
I	clock_mode_t mode;	Clock generator type HIOCLK: High-speed onchip oscillator SYSX1CLK: X1 clock SYSEXTCLK: External main system clock SUBXT1CLK: XT1 clock SUBEXTCLK: External subsystem clock

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)
MD_ERROR2	Exit with error (abend)
MD_ERROR3	Exit with error (abend)
MD_ERROR4	Exit with error (abend)
MD_ARGERROR	Invalid argument specification

R_CGC_Set_CRCOn

Starts the CRC operation function.

[Classification]

r_cg_cgc.c

[Syntax]

```
void R_CGC_Set_CRCOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.2 Port

Below is a list of API functions output by the Code Generator for port use.

Table C-3. API Functions: [Port]

API Function Name	Function
R_PORT_Create	Performs initialization necessary to control port functions.
R_PORT_Create_UserInit	Performs user-defined initialization relating to the port.

R_PORT_Create

Performs initialization necessary to control port functions.

[Classification]

r_cg_port.c

[Syntax]

```
void R_PORT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PORT_Create_UserInit

Performs user-defined initialization relating to the port.

Remark This API function is called as the [R_PORT_Create](#) callback routine.

[Classification]

r_cg_port_user.c

[Syntax]

```
void R_PORT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.3 Interrupt

Below is a list of API functions output by the Code Generator for interrupt and key interrupt use.

Table C-4. API Functions: [Interrupt]

API Function Name	Function
R_INTC_Create	Performs initialization necessary to control the external maskable interrupts $INTP_n$ (and the external maskable interrupts $INTPLR_n$) functions.
R_INTC_Create_UserInit	Performs user-defined initialization relating to the external maskable interrupts $INTP_n$ (and the external maskable interrupts $INTPLR_n$) functions.
r_intcn_interrupt	Performs processing in response to the external maskable interrupt $INTP_n$.
R_INTCn_Start	Enables the acceptance of the external maskable interrupts $INTP_n$.
R_INTCn_Stop	Disables the acceptance of the external maskable interrupts $INTP_n$.
R_KEY_Create	Performs initialization necessary to control the key interrupt $INTKR$ functions.
R_KEY_Create_UserInit	Performs user-defined initialization relating to the key interrupt $INTKR$ functions.
r_key_interrupt	Performs processing in response to the key interrupt $INTKR$.
R_KEY_Start	Enables the acceptance of the key interrupt $INTKR$.
R_KEY_Stop	Disables the acceptance of the key interrupt $INTKR$.
r_intclrn_interrupt	Performs processing in response to the external maskable interrupt $INTPLR_n$.
R_INTCLRn_Start	Enables the acceptance of the external maskable interrupts $INTPLR_n$.
R_INTCLRn_Stop	Disables the acceptance of the external maskable interrupts $INTPLR_n$.

R_INTC_Create

Performs initialization necessary to control the external maskable interrupts $INTP_n$ (and the external maskable interrupts $INTPLR_n$) functions.

[Classification]

r_cg_intc.c

[Syntax]

```
void R_INTC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_INTC_Create_UserInit

Performs user-defined initialization relating to the external maskable interrupts INTP n (and the external maskable interrupts INTPLR n) functions.

Remark This API function is called as the [R_INTC_Create](#) callback routine.

[Classification]

r_cg_intc_user.c

[Syntax]

```
void R_INTC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_intcn_interrupt

Performs processing in response to the external maskable interrupt INTP n .

Remark This API function is called as the interrupt process corresponding to the external maskable interrupt INTP n .

[Classification]

r_cg_intc_user.c

[Syntax]

```
__interrupt static void r_intcn_interrupt ( void );
```

Remark n is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTC n _Start

Enables the acceptance of the external maskable interrupts INTP n .

[Classification]

r_cg_intc.c

[Syntax]

```
void R_INTC $n$ _Start ( void );
```

Remark n is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTC n _Stop

Disables the acceptance of the external maskable interrupts INTP n .

[Classification]

r_cg_intc.c

[Syntax]

```
void R_INTC $n$ _Stop ( void );
```

Remark n is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_KEY_Create

Performs initialization necessary to control the key interrupt INTKR functions.

[Classification]

r_cg_intc.c

[Syntax]

```
void R_KEY_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_KEY_Create_UserInit

Performs user-defined initialization relating to the key interrupt INTKR functions.

Remark This API function is called as the [R_KEY_Create](#) callback routine.

[Classification]

r_cg_intc_user.c

[Syntax]

```
void R_KEY_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_key_interrupt

Performs processing in response to the key interrupt INTKR.

Remark This API function is called as the interrupt process corresponding to the key interrupt INTKR.

[Classification]

r_cg_intc_user.c

[Syntax]

```
__interrupt static void r_key_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_KEY_Start

Enables the acceptance of the key interrupt INTKR.

[Classification]

r_cg_intc.c

[Syntax]

```
void R_KEY_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_KEY_Stop

Disables the acceptance of the key interrupt INTKR.

[Classification]

r_cg_intc.c

[Syntax]

```
void R_KEY_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_intclrn_interrupt

Performs processing in response to the external maskable interrupt INTPLR n .

Remark This API function is called as the interrupt process corresponding to the external maskable interrupt INTPLR n .

[Classification]

r_cg_intc_user.c

[Syntax]

```
__interrupt static void r_intclrn_interrupt ( void );
```

Remark n is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTCLR n _Start

Enables the acceptance of the external maskable interrupts INTPLR n .

[Classification]

r_cg_intc.c

[Syntax]

```
void R_INTCLR $n$ _Start ( void );
```

Remark n is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTCLR n _Stop

Disables the acceptance of the external maskable interrupts INTPLR n .

[Classification]

r_cg_intc.c

[Syntax]

```
void R_INTCLR $n$ _Stop ( void );
```

Remark n is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

C.3.4 Serial

Below is a list of API functions output by the Code Generator for serial array unit and serial interface use.

Table C-5. API Functions: [Serial]

API Function Name	Function
R_SAUm_Create	Performs initialization necessary to control the serial array unit and serial interface functions.
R_SAUm_Create_UserInit	Performs user-defined initialization related to the serial array unit and serial interface functions.
R_SAUm_Set_PowerOff	Halts the clock supplied to the serial array unit.
R_SAUm_Set_SnoozeOn	Enables the switch from STOP mode to SNOOZE mode.
R_SAUm_Set_SnoozeOff	Disables the switch from STOP mode to SNOOZE mode.
R_UARTn_Create	Performs initialization of the serial interface (UART) channel.
r_uartn_interrupt_send	Performs processing in response to the UART transmission end interrupt INTST n .
r_uartn_interrupt_receive	Performs processing in response to the UART reception end interrupt INTSR n .
r_uartn_interrupt_error	Performs processing in response to the reception error interrupt INTSRE n .
R_UARTn_Start	Sets UART communication to standby mode.
R_UARTn_Stop	Ends UART communication.
R_UARTn_Send	Starts UART data transmission.
R_UARTn_Receive	Starts UART data reception.
r_uartn_callback_sendend	Performs processing in response to the UART transmission end interrupt INTST n .
r_uartn_callback_receiveend	Performs processing in response to the UART reception end interrupt INTSR n .
r_uartn_callback_error	Performs processing in response to the UART reception error interrupt INTSRE n .
r_uartn_callback_softwareoverrun	Performs processing in response to detection of overrun error.
R_CSImn_Create	Performs initialization of the serial interface (CSI) channel.
r_csimn_interrupt	Performs processing in response to the CSI communication end interrupt INTCSImn.
R_CSImn_Start	Sets CSI communication to standby mode.
R_CSImn_Stop	Ends CSI communication.
R_CSImn_Send	Starts CSI data transmission.
R_CSImn_Receive	Starts CSI data reception.
R_CSImn_Send_Receive	Starts CSI data transmission/reception.
r_csimn_callback_sendend	Performs processing in response to the CSI transmission end interrupt INTCSImn.
r_csimn_callback_receiveend	Performs processing in response to the CSI reception end interrupt INTCSImn.
r_csimn_callback_error	Performs processing in response to the CSI reception error interrupt INTSRE n .
R_IICmn_Create	Performs initialization of the serial interface (simple IIC) channel.
r_iicmn_interrupt	Performs processing in response to the simple IIC communication end interrupt INTIICmn.

API Function Name	Function
R_IICmn_StartCondition	Generates start conditions.
R_IICmn_StopCondition	Generates stop conditions.
R_IICmn_Stop	Ends simple IIC communication.
R_IICmn_Master_Send	Starts simple IIC master transmission.
R_IICmn_Master_Receive	Starts simple IIC master reception.
r_iicmn_callback_master_sendend	Performs processing in response to the simple IICmn master transmission end interrupt INTIICmn.
r_iicmn_callback_master_receiveend	Performs processing in response to the simple IICmn master reception end interrupt INTIICmn.
r_iicmn_callback_master_error	Performs processing in response to detection of parity error (ACK error).
R_DALIn_Create	Performs initialization of the serial interface (DALI) channel.
r_dalin_interrupt_send	Performs processing in response to the DALI transmission end interrupt INTSTDLn.
r_dalin_interrupt_receive	Performs processing in response to the DALI reception end interrupt INTSRDLn.
r_dalin_interrupt_error	Performs processing in response to the DALI reception error interrupt INTSREDLn.
R_DALIn_Start	Sets DALI communication to standby mode.
R_DALIn_Stop	Ends DALI communication.
R_DALIn_Send	Starts DALI data transmission.
R_DALIn_Receive	Starts DALI data reception.
r_dalin_callback_sendend	Performs processing in response to the DALI transmission end interrupt INTSTDLn.
r_dalin_callback_receiveend	Performs processing in response to the DALI reception end interrupt INTSRDLn.
r_dalin_callback_error	Performs processing in response to the DALI reception error interrupt INTSREDLn.
r_dalin_callback_softwareoverrun	Performs processing in response to detection of overrun error.
R_UARTFn_Create	Performs initialization of the asynchronous serial interface (UARTF) channel.
R_UARTFn_Create_UserInit	Performs user-defined initialization of the asynchronous serial interface (UARTF) channel.
r_uartfn_interrupt_send	Performs processing in response to the LIN-UART transmission end interrupt INTLT.
r_uartfn_interrupt_receive	Performs processing in response to the LIN-UART reception end interrupt INTLR.
r_uartfn_interrupt_error	Performs processing in response to the LIN-UART reception status interrupt INTLS.
R_UARTFn_Start	Sets UARTF communication to standby mode.
R_UARTFn_Stop	Ends UARTF communication.
R_UARTFn_Set_PowerOff	Halts the clock supplied to the asynchronous serial interface (UARTF) channel.
R_UARTFn_Send	Starts UARTF data transmission.

API Function Name	Function
R_UARTFn_Receive	Starts UARTF data reception.
R_UARTFn_Set_DataComparisonOn	Starts the data comparison.
R_UARTFn_Set_DataComparisonOff	Ends the data comparison.
r_uartfn_callback_sendend	Performs processing in response to the LIN-UART transmission end interrupt INTLT.
r_uartfn_callback_receiveend	Performs processing in response to the LIN-UART reception end interrupt INTLR.
r_uartfn_callback_error	Performs processing in response to the LIN-UART reception status interrupt INTLS.
r_uartfn_callback_softwareoverrun	Performs processing in response to detection of overrun error.
r_uartfn_callback_expbitdetect	Performs processing in response to detection of expansion bit.
r_uartfn_callback_idmatch	Performs processing in response to match of ID parity.
R_IICAn_Create	Performs initialization of the serial interface (IICA) channel.
R_IICAn_Create_UserInit	Performs user-defined initialization of the serial interface (IICA) channel.
r_iican_interrupt	Performs processing in response to the IICA communication end interrupt INTIICAn.
R_IICAn_StopCondition	Generates stop conditions.
R_IICAn_Stop	Ends IICA communication.
R_IICAn_Set_PowerOff	Halts the clock supplied to the serial interface (IICA) channel.
R_IICAn_Master_Send	Starts IICA master transmission.
R_IICAn_Master_Receive	Starts IICA master reception.
r_iican_callback_master_sendend	Performs processing in response to the IICA master transmission end interrupt INTIICAn.
r_iican_callback_master_receiveend	Performs processing in response to the IICA master reception end interrupt INTIICAn.
r_iican_callback_master_error	Performs processing in response to detection of IICA master communication error.
R_IICAn_Slave_Send	Starts IICA slave transmission.
R_IICAn_Slave_Receive	Starts IICA slave reception.
r_iican_callback_slave_sendend	Performs processing in response to the IICA slave transmission end interrupt INTIICAn.
r_iican_callback_slave_receiveend	Performs processing in response to the IICA slave reception end interrupt INTIICAn.
r_iican_callback_slave_error	Performs processing in response to detection of IICA slave communication error.
r_iican_callback_getstopcondition	Performs processing in response to detection of stop condition.

R_SAUm_Create

Performs initialization necessary to control the serial array unit and serial interface functions.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_SAUm_Create ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAUm_Create_UserInit

Performs user-defined initialization related to the serial array unit and serial interface functions.

Remark This API function is called as the [R_SAUm_Create](#) callback routine.

[Classification]

r_cg_serial_user.c

[Syntax]

```
void R_SAUm_Create_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAUm_Set_PowerOff

Halts the clock supplied to the serial array unit.

Remark Calling this API function changes the serial array unit to reset status. For this reason, writes to the control registers (e.g. serial clock select register n : SPS n) after this API function is called are ignored.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_SAUm_Set_PowerOff ( void );
```

Remark m is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAUm_Set_SnoozeOn

Enables the switch from STOP mode to SNOOZE mode.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_SAUm_Set_SnoozeOn ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAUm_Set_SnoozeOff

Disables the switch from STOP mode to SNOOZE mode.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_SAUm_Set_SnoozeOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Create

Performs initialization of the serial interface (UART) channel.

Remark This API function is used as an internal function of [R_SAUm_Create](#).
For this reason, there is normally no need to call it from a user program.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_UARTn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartn_interrupt_send

Performs processing in response to the UART transmission end interrupt INTST n .

Remark This API function is called as the interrupt process corresponding to the UART transmission end interrupt INTST n .

[Classification]

r_cg_serial_user.c

[Syntax]

```
__interrupt static void r_uartn_interrupt_send ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartn_interrupt_receive

Performs processing in response to the UART reception end interrupt INTSR n .

Remark This API function is called as the interrupt process corresponding to the UART reception end interrupt INTSR n .

[Classification]

r_cg_serial_user.c

[Syntax]

```
__interrupt static void r_uartn_interrupt_receive ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartn_interrupt_error

Performs processing in response to the reception error interrupt INTSRE n .

Remark This API function is called as the interrupt process corresponding to the reception error interrupt INTSRE n .

[Classification]

r_cg_serial_user.c

[Syntax]

```
__interrupt static void r_uartn_interrupt_error ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Start

Sets UART communication to standby mode.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_UARTn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Stop

Ends UART communication.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_UARTn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTn_Send

Starts UART data transmission.

- Remarks 1.** This API function repeats the byte-level UART transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.
- 2.** When performing a UART transmission, [R_UARTn_Start](#) must be called before this API function is called.

[Classification]

r_cg_serial.c

[Syntax]

```
#include "r_cg_macrodriver.h"
MD_STATUS R_UARTn_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const <i>tx_buf</i> ;	Pointer to a buffer storing the transmission data
I	uint16_t <i>tx_num</i> ;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_UARTn_Receive

Starts UART data reception.

- Remarks 1.** This API function performs byte-level UART reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.
- 2.** Actual UART reception starts after this API function is called, and [R_UARTn_Start](#) is then called.

[Classification]

r_cg_serial.c

[Syntax]

```
#include "r_cg_macrodriver.h"
MD_STATUS R_UARTn_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t * const <i>rx_buf</i> ;	Pointer to a buffer to store the received data
I	uint16_t <i>rx_num</i> ;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

r_uartn_callback_sendend

Performs processing in response to the UART transmission end interrupt INTST n .

Remark This API function is called as the callback routine of interrupt process [r_uartn_interrupt_send](#) corresponding to the UART transmission end interrupt INTST n (performed when number of transmission data specified by [R_UARTn_Send](#) argument tx_num has been completed).

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_uartn_callback_sendend ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartn_callback_receiveend

Performs processing in response to the UART reception end interrupt INTSR n .

Remark This API function is called as the callback routine of interrupt process [r_uartn_interrupt_receive](#) corresponding to the UART reception end interrupt INTSR n (performed when number of received data specified by [R_UARTn_Receive](#) argument rx_num has been completed).

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_uartn_callback_receiveend ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartn_callback_error

Performs processing in response to the UART reception error interrupt INTSRE n .

Remark This API function is called as the callback routine of interrupt process [r_uartn_interrupt_error](#) corresponding to the UART reception error interrupt INTSRE n .

[Classification]

r_cg_serial_user.c

[Syntax]

```
#include "r_cg_macrodriver.h"
static void r_uartn_callback_error ( uint8_t err_type );
```

Remark n is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t <i>err_type</i> ;	Trigger for UART reception error interrupt 00000xx1B: Overrun error 00000x1xB: Parity error 000001xxB: Framing error

[Return value]

None.

r_uartn_callback_softwareoverrun

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process [r_uartn_interrupt_receive](#) corresponding to the UART reception end interrupt INTSR n (process performed when the amount of data received is greater than the argument rx_num specified for [R_UARTn_Receive](#)).

[Classification]

r_cg_serial_user.c

[Syntax]

```
#include "r_cg_macrodriver.h"
static void r_uartn_callback_softwareoverrun ( uint16_t rx_data );
```

Remark n is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint16_t rx_data ;	Receive data (greater than the argument rx_num specified for R_UARTn_Receive)

[Return value]

None.

R_CSImn_Create

Performs initialization of the serial interface (CSI) channel.

Remark This API function is used as an internal function of [R_SAUm_Create](#).
For this reason, there is normally no need to call it from a user program.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_CSImn_Create ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csimn_interrupt

Performs processing in response to the CSI communication end interrupt INTCSImn.

Remark This API function is called as the interrupt process corresponding to the CSI communication end interrupt INTCSImn.

[Classification]

r_cg_serial_user.c

[Syntax]

```
__interrupt static void r_csimn_interrupt ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSImn_Start

Sets CSI communication to standby mode.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_CSImn_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSImn_Stop

Ends CSI communication.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_CSImn_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSImn_Send

Starts CSI data transmission.

- Remarks 1.** This API function repeats the byte-level CSI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.
- 2.** When performing a CSI transmission, [R_CSImn_Start](#) must be called before this API function is called.

[Classification]

r_cg_serial.c

[Syntax]

```
#include "r_cg_macrodriver.h"
MD_STATUS R_CSImn_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const <i>tx_buf</i> ;	Pointer to a buffer storing the transmission data
I	uint16_t <i>tx_num</i> ;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_CSImn_Receive

Starts CSI data reception.

- Remarks 1.** This API function performs byte-level CSI reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.
- 2.** When performing a CSI reception, [R_CSImn_Start](#) must be called before this API function is called.

[Classification]

r_cg_serial.c

[Syntax]

```
#include "r_cg_macrodriver.h"
MD_STATUS R_CSImn_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t * const <i>rx_buf</i> ;	Pointer to a buffer to store the received data
I	uint16_t <i>rx_num</i> ;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_CSImn_Send_Receive

Starts CSI data transmission/reception.

- Remarks 1.** This API function repeats the byte-level CSI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.
- 2.** This API function performs byte-level CSI reception the number of times specified by the argument *tx_num*, and stores the data in the buffer specified by the argument *rx_buf*.
- 3.** When performing a CSI reception, [R_CSImn_Start](#) must be called before this API function is called.

[Classification]

r_cg_serial.c

[Syntax]

```
#include "r_cg_macrodriver.h"
MD_STATUS R_CSImn_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num, uint8_t * const rx_buf );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const <i>tx_buf</i> ;	Pointer to a buffer storing the transmission data
I	uint16_t <i>tx_num</i> ;	Total amount of data to send/receive
O	uint8_t * const <i>rx_buf</i> ;	Pointer to a buffer to store the received data

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

r_csimn_callback_sendend

Performs processing in response to the CSI transmission end interrupt INTCSImn.

Remark This API function is called as the callback routine of interrupt process [r_csimn_interrupt](#) corresponding to the CSI transmission end interrupt INTCSImn (performed when number of transmission data specified by [R_CSImn_Send](#) or [R_CSImn_Send_Receive](#) argument *tx_num* has been completed).

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_csimn_callback_sendend ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csimn_callback_receiveend

Performs processing in response to the CSI reception end interrupt INTCSImn.

Remark This API function is called as the callback routine of interrupt process [r_csimn_interrupt](#) corresponding to the CSI reception end interrupt INTCSImn (performed when number of received data specified by [R_CSImn_Receive](#) or [R_CSImn_Send_Receive](#) argument *rx_num* has been completed).

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_csimn_callback_receiveend ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csimn_callback_error

Performs processing in response to the CSI reception error interrupt INTSRE n .

Remark This API function is called as the callback routine of interrupt process [r_uartn_interrupt_error](#) corresponding to the CSI reception error interrupt INTSRE n .

[Classification]

r_cg_serial_user.c

[Syntax]

```
#include "r_cg_macrodriver.h"
static void r_csimn_callback_error ( uint8_t err_type );
```

Remark m is the unit number, and n is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t <i>err_type</i> ;	Trigger for CSI reception error interrupt 00000xx1B: Overrun error

[Return value]

None.

R_IICmn_Create

Performs initialization of the serial interface (simple IIC) channel.

Remark This API function is used as an internal function of [R_SAUm_Create](#).
For this reason, there is normally no need to call it from a user program.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_IICmn_Create ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iicmn_interrupt

Performs processing in response to the simple IIC communication end interrupt INTIIC*mn*.

Remark This API function is called as the interrupt process corresponding to the simple IIC communication end interrupt INTIIC*mn*.

[Classification]

r_cg_serial_user.c

[Syntax]

```
__interrupt static void r_iicmn_interrupt ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICmn_StartCondition

Generates start conditions.

Remark This API function is used as an internal function of [R_IICmn_Master_Send](#) and [R_IICmn_Master_Receive](#). For this reason, there is normally no need to call it from a user program.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_IICmn_StartCondition ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICmn_StopCondition

Generates stop conditions.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_IICmn_StopCondition ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICmn_Stop

Ends simple IIC communication.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_IICmn_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICmn_Master_Send

Starts simple IIC master transmission.

Remark This API function repeats the byte-level simple IIC master transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

[Classification]

r_cg_serial.c

[Syntax]

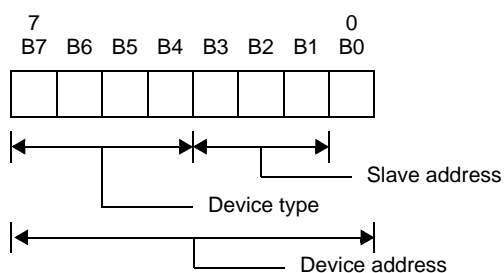
```
#include "r_cg_macrodriver.h"
void R_IICmn_Master_Send ( uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t <i>adr</i> ;	Device address
I	uint8_t * const <i>tx_buf</i> ;	Pointer to a buffer storing the transmission data
I	uint16_t <i>tx_num</i> ;	Total amount of data to send

Remark Below is shown the format for specifying device address *adr*.



[Return value]

None.

R_IICmn_Master_Receive

Starts simple IIC master reception.

Remark This API function performs byte-level simple IIC master reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

[Classification]

r_cg_serial.c

[Syntax]

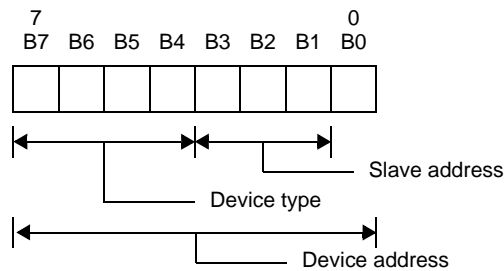
```
#include "r_cg_macrodriver.h"
void R_IICmn_Master_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t <i>adr</i> ;	Device address
O	uint8_t * const <i>rx_buf</i> ;	Pointer to a buffer to store the received data
I	uint16_t <i>rx_num</i> ;	Total amount of data to receive

Remark Below is shown the format for specifying device address *adr*.



[Return value]

None.

r_iicmn_callback_master_sendend

Performs processing in response to the simple IICmn master transmission end interrupt INTIICmn.

Remark This API function is called as the callback routine of interrupt process [r_iicmn_interrupt](#) corresponding to the simple IICmn master transmission end interrupt INTIICmn (performed when number of transmission data specified by [R_IICmn_Master_Send](#) argument *tx_num* has been completed).

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_iicmn_callback_master_sendend ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iicmn_callback_master_receiveend

Performs processing in response to the simple IICmn master reception end interrupt INTIICmn.

Remark This API function is called as the callback routine of interrupt process [r_iicmn_interrupt](#) corresponding to the simple IICmn master reception end interrupt INTIICmn (performed when number of received data specified by [R_IICmn_Master_Receive](#) argument *rx_num* has been completed).

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_iicmn_callback_master_receiveend ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iicmn_callback_master_error

Performs processing in response to detection of parity error (ACK error).

[Classification]

r_cg_serial_user.c

[Syntax]

```
#include "r_cg_macrodriver.h"
static void r_iicmn_callback_master_error ( MD_STATUS flag );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	MD_STATUS <i>flag</i> ;	Cause of communication error MD_NACK: Acknowledge not detected

[Return value]

None.

R_DALIn_Create

Performs initialization of the serial interface (DALI) channel.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_DALIn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dalin_interrupt_send

Performs processing in response to the DALI transmission end interrupt INTSTDL n .

Remark This API function is called as the interrupt process corresponding to the DALI transmission end interrupt INTSTDL n .

[Classification]

r_cg_serial_user.c

[Syntax]

```
__interrupt static void r_dalin_interrupt_send ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dalin_interrupt_receive

Performs processing in response to the DALI reception end interrupt INTSRDL n .

Remark This API function is called as the interrupt process corresponding to the DALI reception end interrupt INTSRDL n .

[Classification]

r_cg_serial_user.c

[Syntax]

```
__interrupt static void r_dalin_interrupt_receive ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dalin_interrupt_error

Performs processing in response to the DALI reception error interrupt INTSREDL n .

Remark This API function is called as the interrupt process corresponding to the DALI reception error interrupt INTSREDL n .

[Classification]

r_cg_serial_user.c

[Syntax]

```
__interrupt static void r_dalin_interrupt_error ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DALIn_Start

Sets DALI communication to standby mode.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_DALIn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DALIn_Stop

Ends DALI communication.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_DALIn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DALIn_Send

Starts DALI data transmission.

- Remarks 1.** This API function repeats the byte-level DALI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.
- 2.** When performing a DALI transmission, [R_DALIn_Start](#) must be called before this API function is called.

[Classification]

r_cg_serial.c

[Syntax]

```
#include "r_cg_macrodriver.h"
MD_STATUS R_DALIn_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const <i>tx_buf</i> ;	Pointer to a buffer storing the transmission data
I	uint16_t <i>tx_num</i> ;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_DALIn_Receive

Starts DALI data reception.

- Remarks 1.** This API function performs byte-level DALI reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.
- 2.** Actual DALI reception starts after this API function is called, and [R_DALIn_Start](#) is then called.

[Classification]

r_cg_serial.c

[Syntax]

```
#include "r_cg_macrodriver.h"
MD_STATUS R_DALIn_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t * const <i>rx_buf</i> ;	Pointer to a buffer to store the received data
I	uint16_t <i>rx_num</i> ;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

r_dalin_callback_sendend

Performs processing in response to the DALI transmission end interrupt INTSTDL n .

Remark This API function is called as the callback routine of interrupt process [r_dalin_interrupt_send](#) corresponding to the DALI transmission end interrupt INTSTDL n (performed when number of transmission data specified by [R_DALIn_Send](#) argument tx_num has been completed).

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_dalin_callback_sendend ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dalin_callback_receiveend

Performs processing in response to the DALI reception end interrupt INTSRDL n .

Remark This API function is called as the callback routine of interrupt process [r_dalin_interrupt_receive](#) corresponding to the DALI reception end interrupt INTSRDL n (performed when number of received data specified by [R_DALIn_Receive](#) argument rx_num has been completed).

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_dalin_callback_receiveend ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dalin_callback_error

Performs processing in response to the DALI reception error interrupt INTSREDL n .

Remark This API function is called as the callback routine of interrupt process [r_dalin_interrupt_error](#) corresponding to the DALI reception error interrupt INTSREDL n .

[Classification]

r_cg_serial_user.c

[Syntax]

```
#include "r_cg_macrodriver.h"
static void r_dalin_callback_error ( uint8_t err_type );
```

Remark n is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t <i>err_type</i> ;	Trigger for DALI reception error interrupt 00000xx1B: Overrun error 00000x1xB: Parity error 000001xxB: Framing error

[Return value]

None.

r_dalin_callback_softwareoverrun

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process [r_dalin_interrupt_receive](#) corresponding to the DALI reception end interrupt INTSRDLn (process performed when the amount of data received is greater than the argument *rx_num* specified for [R_DALIn_Receive](#)).

[Classification]

r_cg_serial_user.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
static void  r_dalin_callback_softwareoverrun ( uint16_t rx_data );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint16_t <i>rx_data</i> ;	Receive data (greater than the argument <i>rx_num</i> specified for R_DALIn_Receive)

[Return value]

None.

R_UARTFn_Create

Performs initialization of the asynchronous serial interface (UARTF) channel.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_UARTFn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTFn_Create_UserInit

Performs user-defined initialization of the asynchronous serial interface (UARTF) channel.

Remark This API function is called as the [R_UARTFn_Create](#) callback routine.

[Classification]

r_cg_serial_user.c

[Syntax]

```
void R_UARTFn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartfn_interrupt_send

Performs processing in response to the LIN-UART transmission end interrupt INTLT.

Remark This API function is called as the interrupt process corresponding to the LIN-UART transmission end interrupt INTLT.

[Classification]

r_cg_serial_user.c

[Syntax]

```
__interrupt static void r_uartfn_interrupt_send ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartfn_interrupt_receive

Performs processing in response to the LIN-UART reception end interrupt INTLR.

Remark This API function is called as the interrupt process corresponding to the LIN-UART reception end interrupt INTLR.

[Classification]

r_cg_serial_user.c

[Syntax]

```
__interrupt static void r_uartfn_interrupt_receive ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartfn_interrupt_error

Performs processing in response to the LIN-UART reception status interrupt INTLS.

Remark This API function is called as the interrupt process corresponding to the LIN-UART reception status interrupt INTLS.

[Classification]

r_cg_serial_user.c

[Syntax]

```
__interrupt static void r_uartfn_interrupt_error ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTFn_Start

Sets UARTF communication to standby mode.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_UARTFn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTFn_Stop

Ends UARTF communication.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_UARTFn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTFn_Set_PowerOff

Halts the clock supplied to the asynchronous serial interface (UARTF) channel.

Remark Calling this API function changes the asynchronous serial interface (UARTF) channel to reset status. For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_UARTFn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTFn_Send

Starts UARTF data transmission.

- Remarks 1.** This API function repeats the byte-level UARTF transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.
- 2.** When performing a UARTF transmission, [R_UARTFn_Start](#) must be called before this API function is called.
- 3.** If the asynchronous serial interface (UARTF) channel is used in expansion bit mode, then store the data to send in the buffer specified by argument *tx_buf*, in the following format.
"8-bit data", "Expansion bit", "8-bit data", "Expansion bit", ...

[Classification]

r_cg_serial.c

[Syntax]

```
MD_STATUS R_UARTFn_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const <i>tx_buf</i> ;	Pointer to a buffer storing the transmission data
I	uint16_t <i>tx_num</i> ;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification
MD_DATAEXISTS	Executing transmission process

R_UARTFn_Receive

Starts UARTF data reception.

- Remarks 1.** This API function performs byte-level UARTF reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.
- 2.** Actual UARTF reception starts after this API function is called, and [R_UARTFn_Start](#) is then called.
- 3.** If the asynchronous serial interface (UARTF) channel is used in expansion bit mode, then the received data is stored in the buffer specified by argument *rx_buf*, in the following format.
"8-bit data", "Expansion bit", "8-bit data", "Expansion bit", ...

[Classification]

r_cg_serial.c

[Syntax]

```
MD_STATUS R_UARTFn_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t * const <i>rx_buf</i> ;	Pointer to a buffer to store the received data
I	uint16_t <i>rx_num</i> ;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_UARTFn_Set_DataComparisonOn

Starts the data comparison.

Remark Calling this API function switches the asynchronous serial interface (UARTF) channel to expansion bit mode (with data comparison).

[Classification]

r_cg_serial.c

[Syntax]

```
void R_UARTFn_Set_DataComparisonOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTFn_Set_DataComparisonOff

Ends the data comparison.

Remark Calling this API function switches the asynchronous serial interface (UARTF) channel to expansion bit mode (with no data comparison).

[Classification]

r_cg_serial.c

[Syntax]

```
void R_UARTFn_Set_DataComparisonOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartfn_callback_sendend

Performs processing in response to the LIN-UART transmission end interrupt INTLT.

Remark This API function is called as the callback routine of interrupt process [r_uartfn_interrupt_send](#) corresponding to the LIN-UART transmission end interrupt INTLT (performed when number of transmission data specified by [R_UARTFn_Send](#) argument *tx_num* has been completed).

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_uartfn_callback_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartfn_callback_receiveend

Performs processing in response to the LIN-UART reception end interrupt INTLR.

Remark This API function is called as the callback routine of interrupt process [r_uartfn_interrupt_receive](#) corresponding to the LIN-UART reception end interrupt INTLR (performed when number of received data specified by [R_UARTFn_Receive](#) argument *rx_num* has been completed).

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_uartfn_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartfn_callback_error

Performs processing in response to the LIN-UART reception status interrupt INTLS.

Remark This API function is called as the callback routine of interrupt process [r_uartfn_interrupt_error](#) corresponding to the LIN-UART reception status interrupt INTLS.

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_uartfn_callback_error ( uint8_t err_type );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t <i>err_type</i> ;	Trigger for LIN-UART reception status interrupt 00000xx1B: Overrun error 00000x1xB: Parity error 000001xxB: Framing error

[Return value]

None.

r_uartfn_callback_softwareoverrun

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process [r_uartfn_interrupt_receive](#) corresponding to the LIN-UART reception end interrupt INTLR (performed when number of received data specified by [R_UARTFn_Receive](#) argument *rx_num* has been completed).

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_uartfn_callback_softwareoverrun ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartfn_callback_expbitdetect

Performs processing in response to detection of expansion bit.

Remark This API function is called as the callback routine of interrupt process [r_uartfn_interrupt_error](#) corresponding to the LIN-UART reception status interrupt INTLS (performed when expansion bit has been detected).

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_uartfn_callback_expbitdetect ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartfn_callback_idmatch

Performs processing in response to match ID parity.

Remark This API function is called as the callback routine of interrupt process [r_uartfn_interrupt_error](#) corresponding to the LIN-UART reception status interrupt INTLS (performed when ID parity has been matched).

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_uartfn_callback_idmatch ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICAn_Create

Performs initialization of the serial interface (IICA) channel.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_IICAn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICAn_Create_UserInit

Performs user-defined initialization of the serial interface (IICA) channel.

Remark This API function is called as the [R_IICAn_Create](#) callback routine.

[Classification]

r_cg_serial_user.c

[Syntax]

```
void R_IICAn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iican_interrupt

Performs processing in response to the IICA communication end interrupt INTIICAn.

Remark This API function is called as the interrupt process corresponding to the IICA communication end interrupt INTIICAn.

[Classification]

r_cg_serial_user.c

[Syntax]

```
__interrupt static void r_iican_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICAn_StopCondition

Generates stop conditions.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_IICAn_StopCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICAn_Stop

Ends IICA communication.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_IICAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICAn_Set_PowerOff

Halts the clock supplied to the serial interface (IICA) channel.

Remark Calling this API function changes the serial interface (IICA) channel to reset status. For this reason, writes to the control registers (e.g. IICA control register n : IICCTL n) after this API function is called are ignored.

[Classification]

r_cg_serial.c

[Syntax]

```
void R_IICAn_Set_PowerOff ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICAn_Master_Send

Starts IICA master transmission.

Remark This API function repeats the byte-level IICA master transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

[Classification]

r_cg_serial.c

[Syntax]

```
#include "r_cg_macrodriver.h"
MD_STATUS R_IICAn_Master_Send ( uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num,
uint8_t wait );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t <i>adr</i> ;	Slave address
I	uint8_t * const <i>tx_buf</i> ;	Pointer to a buffer storing the transmission data
I	uint16_t <i>tx_num</i> ;	Total amount of data to send
I	uint8_t <i>wait</i> ;	Setup time of start conditions

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus communication status
MD_ERROR2	Bus not released status

R_IICAn_Master_Receive

Starts IICA master reception.

Remark This API function performs byte-level IICA master reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

[Classification]

r_cg_serial.c

[Syntax]

```
#include "r_cg_macrodriver.h"
MD_STATUS R_IICAn_Master_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t rx_num,
uint8_t wait );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t <i>adr</i> ;	Slave address
O	uint8_t * const <i>rx_buf</i> ;	Pointer to a buffer to store the received data
I	uint16_t <i>rx_num</i> ;	Total amount of data to receive
I	uint8_t <i>wait</i> ;	Setup time of start conditions

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus communication status
MD_ERROR2	Bus not released status

r_iican_callback_master_sendend

Performs processing in response to the IICA master transmission end interrupt INTIICAn.

Remark This API function is called as the callback routine of interrupt process [r_iican_interrupt](#) corresponding to the IICA master transmission end interrupt INTIICAn.

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_iican_callback_master_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iican_callback_master_receiveend

Performs processing in response to the IICA master reception end interrupt INTIICAn.

Remark This API function is called as the callback routine of interrupt process [r_iican_interrupt](#) corresponding to the IICA master reception end interrupt INTIICAn.

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_iican_callback_master_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iican_callback_master_error

Performs processing in response to detection of IICA master communication error.

[Classification]

r_cg_serial_user.c

[Syntax]

```
#include "r_cg_macrodriver.h"
static void r_iican_callback_master_error ( MD_STATUS flag );
```

Remark n is the channel number.

[Argument(s)]

I/O	Argument	Description
I	MD_STATUS flag;	Cause of communication error MD_SPT: Stop condition detected MD_NACK: Acknowledge not detected

[Return value]

None.

R_IICAn_Slave_Send

Starts IICA slave transmission.

Remark This API function repeats the byte-level IICA slave transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

[Classification]

r_cg_serial.c

[Syntax]

```
#include "r_cg_macrodriver.h"
void R_IICAn_Slave_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const <i>tx_buf</i> ;	Pointer to a buffer storing the transmission data
I	uint16_t <i>tx_num</i> ;	Total amount of data to send

[Return value]

None.

R_IICAn_Slave_Receive

Starts IICA slave reception.

Remark This API function performs byte-level IICA slave reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

[Classification]

r_cg_serial.c

[Syntax]

```
#include "r_cg_macrodriver.h"
void R_IICAn_Slave_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t * const <i>rx_buf</i> ;	Pointer to a buffer to store the received data
I	uint16_t <i>rx_num</i> ;	Total amount of data to receive

[Return value]

None.

r_iican_callback_slave_sendend

Performs processing in response to the IICA slave transmission end interrupt INTIICAn.

Remark This API function is called as the callback routine of interrupt process [r_iican_interrupt](#) corresponding to the IICA slave transmission end interrupt INTIICAn.

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_iican_callback_slave_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iican_callback_slave_receiveend

Performs processing in response to the IICA slave reception end interrupt INTIICAn.

Remark This API function is called as the callback routine of interrupt process [r_iican_interrupt](#) corresponding to the IICA slave reception end interrupt INTIICAn.

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_iican_callback_slave_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iican_callback_slave_error

Performs processing in response to detection of IICA slave communication error.

[Classification]

r_cg_serial_user.c

[Syntax]

```
#include "r_cg_macrodriver.h"
static void r_iican_callback_slave_error ( MD_STATUS flag );
```

Remark n is the channel number.

[Argument(s)]

I/O	Argument	Description
I	MD_STATUS flag;	Cause of communication error MD_ERROR: Address mismatch detected MD_NACK: Acknowledge not detected

[Return value]

None.

r_iican_callback_getstopcondition

Performs processing in response to detection of stop condition.

[Classification]

r_cg_serial_user.c

[Syntax]

```
static void r_iican_callback_getstopcondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.3.5 A/D Converter

Below is a list of API functions output by the Code Generator for A/D converter use.

Table C-6. API Functions: [A/D Converter]

API Function Name	Function
R_ADC_Create	Performs initialization necessary to control A/D converter functions.
R_ADC_Create_UserInit	Performs user-defined initialization relating to the A/D converter.
r_adc_interrupt	Performs processing in response to the A/D conversion end interrupt INTAD.
R_ADC_Set_OperationOn	Enables operation of voltage converter.
R_ADC_Set_OperationOff	Disables operation of voltage converter.
R_ADC_Start	Starts A/D conversion.
R_ADC_Stop	Ends A/D conversion.
R_ADC_Set_PowerOff	Halts the clock supplied to the A/D converter.
R_ADC_Set_ADChannel	Configures the analog voltage input pin for A/D conversion.
R_ADC_Set_SnoozeOn	Enables the switch from STOP mode to SNOOZE mode.
R_ADC_Set_SnoozeOff	Disables the switch from STOP mode to SNOOZE mode.
R_ADC_Set_TestChannel	Sets the operation mode of A/D converter.
R_ADC_Get_Result	Reads the results of A/D conversion (10-bit).
R_ADC_Get_Result_8bit	Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).

R_ADC_Create

Performs initialization necessary to control A/D converter functions.

[Classification]

r_cg_adc.c

[Syntax]

```
void R_ADC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Create_UserInit

Performs user-defined initialization relating to the A/D converter.

Remark This API function is called as the [R_ADC_Create](#) callback routine.

[Classification]

r_cg_adc_user.c

[Syntax]

```
void R_ADC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_adc_interrupt

Performs processing in response to the A/D conversion end interrupt INTAD.

Remark This API function is called as the interrupt process corresponding to the A/D conversion end interrupt INTAD.

[Classification]

r_cg_adc_user.c

[Syntax]

```
__interrupt static void r_adc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_OperationOn

Enables operation of voltage converter.

- Remarks 1.** About 1 microsecond of stabilization time is required when changing the voltage converter from operation stopped to operation enabled status.
Consequently, about 1 micro second must be left free between the call to this API function and the call to [R_ADC_Start](#).
- 2.** On the [A/D Converter], in the [Comparator operation setting] area, if "Operation" is selected, then the voltage converter will be switched to "always on".
There is thus no need to call this API function in this case.

[Classification]

r_cg_adc.c

[Syntax]

```
void R_ADC_Set_OperationOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_OperationOff

Disables operation of voltage converter.

[Classification]

r_cg_adc.c

[Syntax]

```
void R_ADC_Set_OperationOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Start

Starts A/D conversion.

Remark About 1 micro second of stabilization time is required when changing the voltage converter from operation stopped to operation enabled status.
Consequently, about 1 micro second must be left free between the call to [R_ADC_Set_OperationOn](#) and the call to this API function.

[Classification]

r_cg_adc.c

[Syntax]

```
void R_ADC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Stop

Ends A/D conversion.

Remark The voltage converter continues to operate after the process of this API function completes. Consequently, to stop the operation of the voltage converter, you must call [R_ADC_Set_OperationOff](#) after the process of this API function completes.

[Classification]

r_cg_adc.c

[Syntax]

```
void R_ADC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_PowerOff

Halts the clock supplied to the A/D converter.

Remark Calling this API function changes the A/D converter to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_cg_adc.c

[Syntax]

```
void R_ADC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_ADChannel

Configures the analog voltage input pin for A/D conversion.

Remark The value specified in argument *channel* is set to analog input channel specification register (ADS).

[Classification]

r_cg_adc.c

[Syntax]

```
#include "r_cg_macrodriver.h"
#include "r_cg_adc.h"
MD_STATUS R_ADC_Set_ADChannel ( ad_channel_t channel );
```

[Argument(s)]

I/O	Argument	Description
I	ad_channel_t <i>channel</i> ;	Analog voltage input pin ADCHANNEL n : Input pin

Remark See the header file r_cg_adc.h for details about the analog voltage input pin ADCHANNEL n .

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_ADC_Set_SnoozeOn

Enables the switch from STOP mode to SNOOZE mode.

[Classification]

r_cg_adc.c

[Syntax]

```
void R_ADC_Set_SnoozeOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_SnoozeOff

Disables the switch from STOP mode to SNOOZE mode.

[Classification]

r_cg_adc.c

[Syntax]

```
void R_ADC_Set_SnoozeOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_TestChannel

Sets the operation mode of A/D converter.

[Classification]

r_cg_adc.c

[Syntax]

```
#include "r_cg_macrodriver.h"
#include "r_cg_adc.h"
MD_STATUS R_ADC_Set_TestChannel ( test_channel_t channel );
```

[Argument(s)]

I/O	Argument	Description
I	test_channel_t channel;	Operation mode of A/D converter ADNORMALINPUT : Normal mode (Normal A/D conversion) ADAVREFM : Test mode (AVREFM input) ADAVREFP : Test mode (AVREFP input)

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_ADC_Get_Result

Reads the results of A/D conversion (10-bit).

[Classification]

r_cg_adc.c

[Syntax]

```
#include "r_cg_macrodriver.h"
void R_ADC_Get_Result ( uint16_t * const buffer );
```

[Argument(s)]

I/O	Argument	Description
O	uint16_t * const <i>buffer</i> ;	Pointer to area in which to store read results of A/D conversion

[Return value]

None.

R_ADC_Get_Result_8bit

Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).

[Classification]

r_cg_adc.c

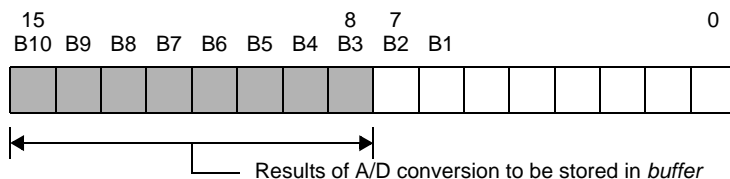
[Syntax]

```
#include "r_cg_macrodriver.h"
void R_ADC_Get_Result_8bit ( uint8_t * const buffer );
```

[Argument(s)]

I/O	Argument	Description
O	uint8_t * const <i>buffer</i> ;	Pointer to area in which to store the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution)

Remark Below is an example of the results of A/D conversion to be stored in *buffer*.



[Return value]

None.

C.3.6 D/A Converter

Below is a list of API functions output by the Code Generator for D/A converter use.

Table C-7. API Functions: [D/A Converter]

API Function Name	Function
R_DAC_Create	Performs initialization necessary to control the D/A converter functions.
R_DAC_Create_UserInit	Performs user-defined initialization relating to the D/A converter.
R_DACn_Start	Starts D/A conversion.
R_DACn_Stop	Ends D/A conversion.
R_DAC_Set_PowerOff	Halts the clock supplied to the D/A converter.
R_DACn_Set_ConversionValue	Sets the analog voltage output to the ANOn pin.

R_DAC_Create

Performs initialization necessary to control the D/A converter functions.

[Classification]

r_cg_dac.c

[Syntax]

```
void R_DAC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DAC_Create_UserInit

Performs user-defined initialization relating to the D/A converter.

Remark This API function is called as the [R_DAC_Create](#) callback routine.

[Classification]

r_cg_dac_user.c

[Syntax]

```
void R_DAC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DAC n _Start

Starts D/A conversion.

[Classification]

r_cg_dac.c

[Syntax]

```
void R_DACn_Start ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DACn_Stop

Ends D/A conversion.

[Classification]

r_cg_dac.c

[Syntax]

```
void R_DACn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DAC_Set_PowerOff

Halts the clock supplied to the D/A converter.

Remark Calling this API function changes the D/A converter to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_cg_dac.c

[Syntax]

```
void R_DAC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DACn_Set_ConversionValue

Sets the analog voltage output to the ANOn pin.

[Classification]

r_cg_dac.c

[Syntax]

```
#include "r_cg_macrodriver.h"
void R_DACn_Set_ConversionValue ( uint8_t reg_value );
```

Remark n is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t reg_value;	D/A conversion value (0x0 to 0xFF)

[Return value]

None.

C.3.7 Timer

Below is a list of API functions output by the Code Generator for timer array unit and timer use.

Table C-8. API Functions: [Timer]

API Function Name	Function
R_TAUm_Create	Performs initialization necessary to control timer array unit functions.
R_TAUm_Create_UserInit	Performs user-defined initialization relating to the timer array unit.
r_taum_channeln_interrupt	Performs processing in response to the timer interrupt INTT Mmn .
r_taum_channeln_higher8bits_interrupt	Performs processing in response to the timer interrupt INTT $MmnH$.
R_TAUm_Channeln_Start	Starts the count for channel n .
R_TAUm_Channeln_Higher8bits_Start	Starts the count (higher 8-bit) for channel 1 or channel 3.
R_TAUm_Channeln_Lower8bits_Start	Starts the count (lower 8-bit) for channel 1 or channel 3.
R_TAUm_Channeln_Stop	Ends the count for channel n .
R_TAUm_Channeln_Higher8bits_Stop	Ends the count (higher 8-bit) for channel 1 or channel 3.
R_TAUm_Channeln_Lower8bits_Stop	Ends the count (lower 8-bit) for channel 1 or channel 3.
R_TAUm_Set_PowerOff	Halts the clock supplied to the timer array unit.
R_TAUm_Channeln_Get_PulseWidth	Captures the high/low-level width measured between pulses of the signal (pulses) input to the T lmn pin.
R_TAUm_Channeln_Set_SoftwareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.
R_TMR_RDn_Create	Performs initialization necessary to control the 16-bit timer RD n functions.
R_TMR_RDn_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RD n .
r_tmr_rdn_interrupt	Performs processing in response to the timer interrupt.
R_TMR_RDn_Start	Starts the count for 16-bit timer RD n .
R_TMR_RDn_Stop	Ends the count for 16-bit timer RD n .
R_TMR_RDn_Set_PowerOff	Halts the clock supplied to the 16-bit timer RD n .
R_TMR_RDn_ForcedOutput_Start	Starts the pulse output forced cutoff for 16-bit timer RD n .
R_TMR_RDn_ForcedOutput_Stop	Ends the pulse output forced cutoff for 16-bit timer RD n .
R_TMR_RDn_Get_PulseWidth	Reads the pulse width of the 16-bit timer RD n .
R_TMR_RG0_Create	Performs initialization necessary to control the 16-bit timer RG0 functions.
R_TMR_RG0_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RG0.
r_tmr_rg0_interrupt	Performs processing in response to the timer interrupt.
R_TMR_RG0_Start	Starts the count for 16-bit timer RG0.
R_TMR_RG0_Stop	Ends the count for 16-bit timer RG0.
R_TMR_RG0_Set_PowerOff	Halts the clock supplied to the 16-bit timer RG0.
R_TMR_RG0_Get_PulseWidth	Reads the pulse width of the 16-bit timer RG0.
R_TMR_RJ0_Create	Performs initialization necessary to control the 16-bit timer RJ0 functions.
R_TMR_RJ0_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RJ0.
r_tmr_rj0_interrupt	Performs processing in response to the timer interrupt.
R_TMR_RJ0_Start	Starts the count for 16-bit timer RJ0.

API Function Name	Function
R_TMR_RJ0_Stop	Ends the count for 16-bit timer RJ0.
R_TMR_RJ0_Set_PowerOff	Halts the clock supplied to the 16-bit timer RJ0.
R_TMR_RJ0_Get_PulseWidth	Reads the pulse width of the 16-bit timer RJ0.
R_TMR_KB_Create	Performs initialization necessary to control the 16-bit timer KB <i>m</i> functions.
R_TMR_KBm_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer KB <i>m</i> .
r_tmr_kbm_interrupt	Performs processing in response to the timer interrupt.
R_TMR_KBm_Start	Starts the count for 16-bit timer KB <i>m</i> .
R_TMR_KBm_Stop	Ends the count for 16-bit timer KB <i>m</i> .
R_TMR_KBm_Set_PowerOff	Halts the clock supplied to the 16-bit timer KB <i>m</i> .
R_TMR_KBm_ForcedOutput_Start	Enables input of the trigger signal used for the forced output stop function.
R_TMR_KBm_ForcedOutput_Stop	Disables input of the trigger signal used for the forced output stop function.
R_TMR_KC0_Create	Performs initialization necessary to control the 16-bit timer KC0 functions.
R_TMR_KC0_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer KC0.
r_tmr_kc0_interrupt	Performs processing in response to the timer interrupt.
R_TMR_KC0_Start	Starts the count for 16-bit timer KC0.
R_TMR_KC0_Stop	Ends the count for 16-bit timer KC0.
R_TMR_KC0_Set_PowerOff	Halts the clock supplied to the 16-bit timer KC0.
R_WUTM_Create	Performs initialization necessary to control the 16-bit wakeup timer WUTM functions.
R_WUTM_Create_UserInit	Performs user-defined initialization relating to the 16-bit wakeup timer WUTM.
r_wutm_interrupt	Performs processing in response to the timer interrupt.
R_WUTM_Start	Starts the count for 16-bit wakeup timer WUTM.
R_WUTM_Stop	Ends the count for 16-bit wakeup timer WUTM.
R_WUTM_Set_PowerOff	Halts the clock supplied to the 16-bit wakeup timer WUTM.

R_TAUm_Create

Performs initialization necessary to control timer array unit functions.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TAUm_Create ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAUm_Create_UserInit

Performs user-defined initialization relating to the timer array unit.

Remark This API function is called as the [R_TAUm_Create](#) callback routine.

[Classification]

r_cg_timer_user.c

[Syntax]

```
void R_TAUm_Create_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_taum_channel*n*_interrupt

Performs processing in response to the timer interrupt INTT*Mn*.

Remark This API function is called as the interrupt process corresponding to the timer interrupt INTT*Mn*.

[Classification]

r_cg_timer_user.c

[Syntax]

```
__interrupt static void r_taum_channeln_interrupt ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_taum_channeln_higher8bits_interrupt

Performs processing in response to the timer interrupt INTTM m nH.

Remark This API function is called as the interrupt process corresponding to the timer interrupt INTTM m nH.

[Classification]

r_cg_timer_user.c

[Syntax]

```
__interrupt static void r_taum_channeln_higher8bits_interrupt ( void );
```

Remark m is the unit number, and n is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAUm_Channeln_Start

Starts the count for channel n .

Remark The time from the call to this API function to the start of counting depends on the type of the function in question (e.g. interval timer, square-wave output, or external event counter).

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TAUm_Channeln_Start ( void );
```

Remark m is the unit number, and n is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAUm_Channel*n*_Higher8bits_Start

Starts the count (higher 8-bit) for channel 1 or channel 3.

Remark This API function can only be called when the timer array unit is used as a 8-bit timer.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TAUm_Channeln_Higher8bits_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAUm_Channel*n*_Lower8bits_Start

Starts the count (lower 8-bit) for channel 1 or channel 3.

- Remarks 1.** This API function can only be called when the timer array unit is used as a 8-bit timer.
- 2.** The time from the call to this API function to the start of counting depends on the type of the function in question (e.g. interval timer, external event counter, or delay counter).

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TAUm_Channeln_Lower8bits_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAUm_Channeln_Stop

Ends the count for channel *n*.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TAUm_Channeln_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAUm_Channel*n*_Higher8bits_Stop

Ends the count (higher 8-bit) for channel 1 or channel 3.

Remark This API function can only be called when the timer array unit is used as a 8-bit timer.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TAUm_Channeln_Higher8bits_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAUm_Channel*n*_Lower8bits_Stop

Ends the count (lower 8-bit) for channel 1 or channel 3.

Remark This API function can only be called when the timer array unit is used as a 8-bit timer.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TAUm_Channeln_Lower8bits_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAUm_Set_PowerOff

Halts the clock supplied to the timer array unit.

Remark Calling this API function changes the timer array unit to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TAUm_Set_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAUm_Channeln_Get_PulseWidth

Captures the high/low-level width measured between pulses of the signal (pulses) input to the TImn pin.

[Classification]

r_cg_timer.c

[Syntax]

```
#include "r_cg_macrodriver.h"
void R_TAUm_Channeln_Get_PulseWidth ( uint32_t * const width );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	<code>uint32_t * const width;</code>	Pointer to an area to store the measurement width (0x0 to 0x1FFFF)

[Return value]

None.

R_TAU m _Channel n _Set_SoftwareTriggerOn

Generates the trigger (software trigger) for one-shot pulse output.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TAU $m$ _Channel $n$ _Set_SoftwareTriggerOn ( void );
```

Remark m is the unit number, and n is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RDn_Create

Performs initialization necessary to control the 16-bit timer RD*n* functions.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RDn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RDn_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RD*n*.

Remark This API function is called as the [R_TMR_RDn_Create](#) callback routine.

[Classification]

r_cg_timer_user.c

[Syntax]

```
void R_TMR_RDn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tmr_rdn_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Classification]

r_cg_timer_user.c

[Syntax]

```
__interrupt static void r_tmr_rdn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RDn_Start

Starts the count for 16-bit timer RD*n*.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RDn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RDn_Stop

Ends the count for 16-bit timer RD*n*.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RDn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RDn_Set_PowerOff

Halts the clock supplied to the 16-bit timer RD n .

Remark Calling this API function changes the 16-bit timer RD n to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RDn_Set_PowerOff ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RDn_ForcedOutput_Start

Starts the pulse output forced cutoff for 16-bit timer RD*n*.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RDn_ForcedOutput_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RDn_ForcedOutput_Stop

Ends the pulse output forced cutoff for 16-bit timer RD*n*.

Remark This API function can only be called when the 16-bit timer RD*n* is the count to stopped (the TSTART*i* bit in the timer RD start register (TRDSTR) is 0).

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RDn_ForcedOutput_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RDn_Get_PulseWidth

Reads the pulse width of the 16-bit timer RDn.

- Remarks**
1. This API function can only be called when the 16-bit timer RDn is being used for input capture function.
 2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Classification]

r_cg_timer.c

[Syntax]

```
#include "r_cg_macrodriver.h"
#include "r_cg_timer.h"
MD_STATUS R_TMR_RDn_Get_PulseWidth ( uint32_t * const active_width, uint32_t * const
inactive_width, timer_channel_t channel );
```

Remark n is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint32_t * const active_width;	Pointer to an area storing the active level width that was read
O	uint32_t * const inactive_width;	Pointer to an area storing the inactive level width that was read
I	timer_channel_t channel;	Pin to read TMCHANNELA: TRDIOAn pin TMCHANNELB: TRDIOBn pin TMCHANNELC: TRDIOCn pin TMCHANNELD: TRDIODn pin

[Return value]

Macro	Description
MD_OK	Normal completion

R_TMR_RG0_Create

Performs initialization necessary to control the 16-bit timer RG0 functions.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RG0_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RG0.

Remark This API function is called as the [R_TMR_RG0_Create](#) callback routine.

[Classification]

r_cg_timer_user.c

[Syntax]

```
void R_TMR_RG0_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tmr_rg0_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Classification]

r_cg_timer_user.c

[Syntax]

```
__interrupt static void r_tmr_rg0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Start

Starts the count for 16-bit timer RG0.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RG0_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Stop

Ends the count for 16-bit timer RG0.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RG0_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Set_PowerOff

Halts the clock supplied to the 16-bit timer RG0.

Remark Calling this API function changes the 16-bit timer RG0 to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RG0_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Get_PulseWidth

Reads the pulse width of the 16-bit timer RG0.

- Remarks 1.** This API function can only be called when the 16-bit timer RG0 is being used for input capture function.
- 2.** If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Classification]

r_cg_timer.c

[Syntax]

```
#include "r_cg_macrodriver.h"
#include "r_cg_timer.h"
MD_STATUS R_TMR_RG0_Get_PulseWidth ( uint32_t * const active_width, uint32_t * const inactive_width, timer_channel_t channel );
```

[Argument(s)]

I/O	Argument	Description
O	uint32_t * const <i>active_width</i> ;	Pointer to an area storing the active level width that was read from the TRGIOA pin
O	uint32_t * const <i>inactive_width</i> ;	Pointer to an area storing the inactive level width that was read from the TRGIOA pin
I	timer_channel_t <i>channel</i> ;	Pin to read TMCHANNELA: TRGIOA0 pin TMCHANNELB: TRGIOB0 pin

[Return value]

Macro	Description
MD_OK	Normal completion

R_TMR_RJ0_Create

Performs initialization necessary to control the 16-bit timer RJ0 functions.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RJ0_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RJ0.

Remark This API function is called as the [R_TMR_RJ0_Create](#) callback routine.

[Classification]

r_cg_timer_user.c

[Syntax]

```
void R_TMR_RJ0_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tmr_rj0_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Classification]

r_cg_timer_user.c

[Syntax]

```
__interrupt static void r_tmr_rj0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Start

Starts the count for 16-bit timer RJ0.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RJ0_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Stop

Ends the count for 16-bit timer RJ0.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RJ0_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Set_PowerOff

Halts the clock supplied to the 16-bit timer RJ0.

Remark Calling this API function changes the 16-bit timer RJ0 to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_RJ0_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Get_PulseWidth

Reads the pulse width of the 16-bit timer RJ0.

- Remarks 1.** This API function can only be called when the 16-bit timer RJ0 is being used for pulse width measurement mode / pulse period measurement mode.
- 2.** If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Classification]

r_cg_timer.c

[Syntax]

```
#include "r_cg_macrodriver.h"
void R_TMR_RJ0_Get_PulseWidth ( uint32_t * const active_width );
```

[Argument(s)]

I/O	Argument	Description
O	<code>uint32_t * const active_width;</code>	Pointer to an area storing the active level width that was read from the TRJ0IO pin

[Return value]

None.

R_TMR_KB_Create

Performs initialization necessary to control the 16-bit timer KB*m* functions.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_KB_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KBm_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer KB*m*.

Remark This API function is called as the [R_TMR_KB_Create](#) callback routine.

[Classification]

r_cg_timer_user.c

[Syntax]

```
void R_TMR_KBm_Create_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_tmr_kbm_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Classification]

r_cg_timer_user.c

[Syntax]

```
__interrupt static void r_tmr_kbm_interrupt ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KBm_Start

Starts the count for 16-bit timer KB*m*.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_KBm_Start ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KBm_Stop

Ends the count for 16-bit timer KB*m*.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_KBm_Stop ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KBm_Set_PowerOff

Halts the clock supplied to the 16-bit timer KB*m*.

Remark Calling this API function changes the 16-bit timer KB*m* to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_KBm_Set_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KBm_ForcedOutput_Start

Enables input of the trigger signal used for the forced output stop function.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_KBm_ForcedOutput_Start ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KBm_ForcedOutput_Stop

Disables input of the trigger signal used for the forced output stop function.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_KBm_ForcedOutput_Stop ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KC0_Create

Performs initialization necessary to control the 16-bit timer KC0 functions.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_KC0_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KC0_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer KC0.

Remark This API function is called as the [R_TMR_KC0_Create](#) callback routine.

[Classification]

r_cg_timer_user.c

[Syntax]

```
void R_TMR_KC0_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tmr_kc0_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Classification]

r_cg_timer_user.c

[Syntax]

```
__interrupt static void r_tmr_kc0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KC0_Start

Starts the count for 16-bit timer KC0.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_KC0_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KC0_Stop

Ends the count for 16-bit timer KC0.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_KC0_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KC0_Set_PowerOff

Halts the clock supplied to the 16-bit timer KC0.

Remark Calling this API function changes the 16-bit timer KC0 to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_TMR_KC0_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WUTM_Create

Performs initialization necessary to control the 16-bit wakeup timer WUTM functions.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_WUTM_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WUTM_Create_UserInit

Performs user-defined initialization relating to the 16-bit wakeup timer WUTM.

Remark This API function is called as the [R_WUTM_Create](#) callback routine.

[Classification]

r_cg_timer_user.c

[Syntax]

```
void R_WUTM_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_wutm_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Classification]

r_cg_timer_user.c

[Syntax]

```
__interrupt static void r_wutm_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WUTM_Start

Starts the count for 16-bit wakeup timer WUTM.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_WUTM_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WUTM_Stop

Ends the count for 16-bit wakeup timer WUTM.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_WUTM_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WUTM_Set_PowerOff

Halts the clock supplied to the 16-bit wakeup timer WUTM.

Remark Calling this API function changes the 16-bit wakeup timer WUTM to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_cg_timer.c

[Syntax]

```
void R_WUTM_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.8 Watchdog Timer

Below is a list of API functions output by the Code Generator for watchdog timer use.

Table C-9. API Functions: [Watchdog Timer]

API Function Name	Function
R_WDT_Create	Performs initialization necessary to control watchdog timer functions.
R_WDT_Create_UserInit	Performs user-defined initialization relating to the watchdog timer.
r_wdt_interrupt	Performs processing in response to the interval interrupt INTWDTI.
R_WDT_Restart	Clears the watchdog timer counter and resumes counting.

R_WDT_Create

Performs initialization necessary to control watchdog timer functions.

[Classification]

r_cg_wdt.c

[Syntax]

```
void R_WDT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WDT_Create_UserInit

Performs user-defined initialization relating to the watchdog timer.

Remark This API function is called as the [R_WDT_Create](#) callback routine.

[Classification]

r_cg_wdt_user.c

[Syntax]

```
void R_WDT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_wdt_interrupt

Performs processing in response to the interval interrupt INTWDTI.

Remark This API function is called as the interrupt process corresponding to the interval interrupt INTWDTI.

[Classification]

r_cg_wdt_user.c

[Syntax]

```
__interrupt static void r_wdt_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WDT_Restart

Clears the watchdog timer counter and resumes counting.

[Classification]

r_cg_wdt.c

[Syntax]

```
void R_WDT_Restart ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.9 Real-time Clock

Below is a list of API functions output by the Code Generator for real-time clock use.

Table C-10. API Functions: [Real-time Clock]

API Function Name	Function
R_RTC_Create	Performs initialization necessary to control real-time clock functions.
R_RTC_Create_UserInit	Performs user-defined initialization relating to the real-time clock.
r_rtc_interrupt	Performs processing in response to the real-time clock interrupt INTRTC.
R_RTC_Start	Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second).
R_RTC_Stop	Ends the count of the real-time clock (year, month, weekday, day, hour, minute, second).
R_RTC_Set_PowerOff	Halts the clock supplied to the real-time clock.
R_RTC_Set_HourSystem	Sets the clock type (12-hour or 24-hour clock) of the real-time clock.
R_RTC_Set_CounterValue	Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.
R_RTC_Get_CounterValue	Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.
R_RTC_Set_ConstPeriodInterruptOn	Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.
R_RTC_Set_ConstPeriodInterruptOff	Ends the cyclic interrupt function.
r_rtc_callback_constperiod	Performs processing in response to the cyclic interrupt INTRTC.
R_RTC_Set_AlarmOn	Starts the alarm interrupt function.
R_RTC_Set_AlarmOff	Ends the alarm interrupt function.
R_RTC_Set_AlarmValue	Sets the alarm conditions (weekday, hour, minute).
R_RTC_Get_AlarmValue	Reads the alarm conditions (weekday, hour, minute).
r_rtc_callback_alarm	Performs processing in response to the alarm interrupt INTRTC.
R_RTC_Set_RTC1HZOn	Enables output of the correction clock (1 Hz) to the RTC1HZ pin.
R_RTC_Set_RTC1HZOff	Disables output of the correction clock (1 Hz) to the RTC1HZ pin.

R_RTC_Create

Performs initialization necessary to control real-time clock functions.

[Classification]

r_cg_rtc.c

[Syntax]

```
void R_RTC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Create_UserInit

Performs user-defined initialization relating to the real-time clock.

Remark This API function is called as the [R_RTC_Create](#) callback routine.

[Classification]

r_cg_rtc_user.c

[Syntax]

```
void R_RTC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_interrupt

Performs processing in response to the real-time clock interrupt INTRTC.

Remark This API function is called as the interrupt process corresponding to the real-time clock interrupt INTRTC.

[Classification]

r_cg_rtc_user.c

[Syntax]

```
__interrupt static void r_rtc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Start

Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second).

[Classification]

r_cg_rtc.c

[Syntax]

```
void R_RTC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Stop

Ends the count of the real-time clock (year, month, weekday, day, hour, minute, second).

[Classification]

r_cg_rtc.c

[Syntax]

```
void R_RTC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_PowerOff

Halts the clock supplied to the real-time clock.

- Remarks 1.** Calling this API function changes the real-time clock to reset status.
For this reason, writes to the control registers after this API function is called are ignored.
- 2.** This API function stops the clock supply to the real-time clock, by operating the RTCEN bit of peripheral enable register *n*.
For this reason, this API function also stops the clock supply to other peripheral devices sharing the RTCEN bit (e.g. interval timer).

[Classification]

r_cg_rtc.c

[Syntax]

```
void R_RTC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_HourSystem

Sets the clock type (12-hour or 24-hour clock) of the real-time clock.

[Classification]

r_cg_rtc.c

[Syntax]

```
#include "r_cg_macrodriver.h"
#include "r_cg_rtc.h"
MD_STATUS R_RTC_Set_HourSystem ( rtc_hour_system_t hour_system );
```

[Argument(s)]

I/O	Argument	Description
I	<i>rtc_hour_system_t hour_system;</i>	Clock type HOUR12: 12-hour clock HOUR24: 24-hour clock

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)
MD_BUSY2	Stopping count process (after change to setting)
MD_ARGERROR	Invalid argument specification

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Set_CounterValue

Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.

[Classification]

r_cg_rtc.c

[Syntax]

```
#include "r_cg_macrodriver.h"
#include "r_cg_rtc.h"
MD_STATUS R_RTC_Set_CounterValue ( rtc_counter_value_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	rtc_counter_value_t counter_write_val;	Counter value

Remark Below is an example of the structure rtc_counter_value_t (counter value) for the real-time clock.

```
typedef struct {
    uint8_t sec; /* second */
    uint8_t min; /* Minute */
    uint8_t hour; /* Hour */
    uint8_t day; /* Day */
    uint8_t week; /* Weekday (0: Sunday, 6: Saturday) */
    uint8_t month; /* Month */
    uint8_t year; /* Year */
} rtc_counter_value_t;
```

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)
MD_BUSY2	Stopping count process (after change to setting)

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Get_CounterValue

Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.

[Classification]

r_cg_rtc.c

[Syntax]

```
#include "r_cg_macrodriver.h"
#include "r_cg_rtc.h"
MD_STATUS R_RTC_Get_CounterValue ( rtc_counter_value_t * const counter_read_val );
```

[Argument(s)]

I/O	Argument	Description
O	rtc_counter_value_t * const counter_read_val;	Pointer to structure in which to store the counter value being read

Remark See [R_RTC_Set_CounterValue](#) for details about the rtc_counter_value_t counter value.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before reading)
MD_BUSY2	Stopping count process (after reading)

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Set_ConstPeriodInterruptOn

Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.

[Classification]

r_cg_rtc.c

[Syntax]

```
#include "r_cg_macrodriver.h"
#include "r_cg_rtc.h"
MD_STATUS R_RTC_Set_ConstPeriodInterruptOn ( rtc_int_period_t period );
```

[Argument(s)]

I/O	Argument	Description
I	rtc_int_period_t <i>period</i> ;	Interrupt INTRTC cycle HALFSEC: 0.5 seconds ONESEC: 1 second ONEMIN: 1 minute ONEHOUR: 1 hour ONEDAY: 1 day ONEMONTH: 1 month

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_RTC_Set_ConstPeriodInterruptOff

Ends the cyclic interrupt function.

[Classification]

r_cg_rtc.c

[Syntax]

```
void R_RTC_Set_ConstPeriodInterruptOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_callback_constperiod

Performs processing in response to the cyclic interrupt INTRTC.

Remark This API function is called as the callback routine of interrupt process [r_rtc_interrupt](#) corresponding to the cyclic interrupt INTRTC.

[Classification]

r_cg_rtc_user.c

[Syntax]

```
static void r_rtc_callback_constperiod ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_AlarmOn

Starts the alarm interrupt function.

[Classification]

r_cg_rtc.c

[Syntax]

```
void R_RTC_Set_AlarmOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_AlarmOff

Ends the alarm interrupt function.

[Classification]

r_cg_rtc.c

[Syntax]

```
void R_RTC_Set_AlarmOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_AlarmValue

Sets the alarm conditions (weekday, hour, minute).

[Classification]

r_cg_rtc.c

[Syntax]

```
#include "r_cg_macrodriver.h"
#include "r_cg_rtc.h"
void R_RTC_Set_AlarmValue ( rtc_alarm_value_t alarm_val );
```

[Argument(s)]

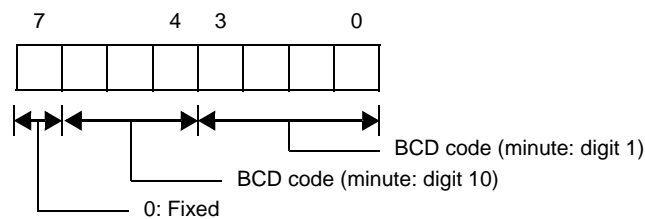
I/O	Argument	Description
I	rtc_alarm_value_t alarm_val;	Alarm conditions (weekday, hour, minute)

Remark Below is shown the structure rtc_alarm_value_t (alarm conditions).

```
typedef struct {
    uint8_t alarmwm; /* Minute */
    uint8_t alarmwh; /* Hour */
    uint8_t alarmmw; /* Weekday */
} rtc_alarm_value_t;
```

- alarmwm (Minute)

Below are shown the meanings of each bit of the structure member alarmwm.

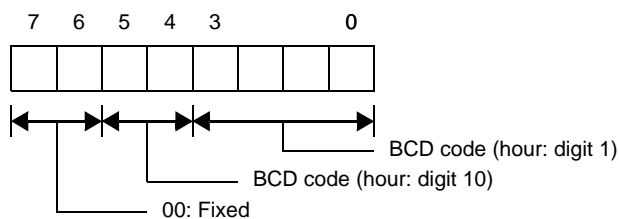


- alarmwh (Hour)

Below are shown the meanings of each bit of the structure member alarmwh.

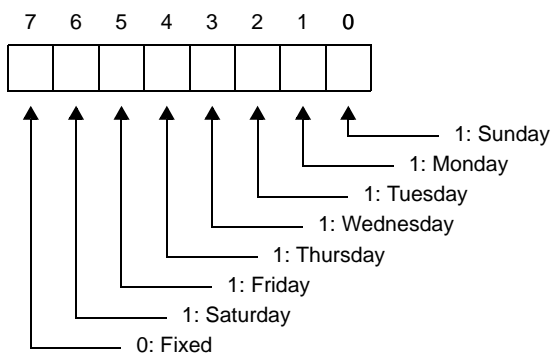
If the real-time clock is set to the 12-hour clock, then bit 5 has the following meaning.

- 0: AM
- 1: PM



- alarmww (Weekday)

Below are shown the meanings of each bit of the structure member alarmww.



[Return value]

None.

R_RTC_Get_AlarmValue

Reads the alarm conditions (weekday, hour, minute).

[Classification]

r_cg_rtc.c

[Syntax]

```
#include "r_cg_macrodriver.h"
#include "r_cg_rtc.h"
void R_RTC_Get_AlarmValue ( rtc_alarm_value_t * const alarm_val );
```

Remark See [R_RTC_Set_AlarmValue](#) for details about rtc_alarm_value_t (alarm conditions).

[Argument(s)]

I/O	Argument	Description
O	rtc_alarm_value_t * const alarm_val;	Pointer to structure in which to store the conditions being read

[Return value]

None.

r_rtc_callback_alarm

Performs processing in response to the alarm interrupt INTRTC.

Remark This API function is called as the callback routine of interrupt process [r_rtc_interrupt](#) corresponding to the alarm interrupt INTRTC.

[Classification]

r_cg_rtc_user.c

[Syntax]

```
static void r_rtc_callback_alarm ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTC1HZOn

Enables output of the correction clock (1 Hz) to the RTC1HZ pin.

[Classification]

r_cg_rtc.c

[Syntax]

```
void R_RTC_Set_RTC1HZOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTC1HZOff

Disables output of the correction clock (1 Hz) to the RTC1HZ pin.

[Classification]

r_cg_rtc.c

[Syntax]

```
void R_RTC_Set_RTC1HZOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.10 Interval Timer

Below is a list of API functions output by the Code Generator for interval timer use.

Table C-11. API Functions: [Interval Timer]

API Function Name	Function
R_IT_Create	Performs initialization necessary to control interval timer functions.
R_IT_Create_UserInit	Performs user-defined initialization relating to the interval timer.
r_it_interrupt	Performs processing in response to the interval timer interrupt INTIT.
R_IT_Start	Starts the count of the interval timer.
R_IT_Stop	Ends the count of the interval timer.
R_IT_Set_PowerOff	Halts the clock supplied to the interval timer.

R_IT_Create

Performs initialization necessary to control interval timer functions.

[Classification]

r_cg_it.c

[Syntax]

```
void R_IT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Create_UserInit

Performs user-defined initialization relating to the interval timer.

Remark This API function is called as the [R_IT_Create](#) callback routine.

[Classification]

r_cg_it_user.c

[Syntax]

```
void R_IT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_it_interrupt

Performs processing in response to the interval timer interrupt INTIT.

Remark This API function is called as the interrupt process corresponding to the interval timer interrupt INTIT.

[Classification]

r_cg_it_user.c

[Syntax]

```
__interrupt static void r_it_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Start

Starts the count of the interval timer.

[Classification]

r_cg_it.c

[Syntax]

```
void R_IT_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Stop

Ends the count of the interval timer.

[Classification]

r_cg_it.c

[Syntax]

```
void R_IT_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Set_PowerOff

Halts the clock supplied to the interval timer.

- Remarks 1.** Calling this API function changes the interval timer to reset status.
For this reason, writes to the control registers after this API function is called are ignored.
- 2.** This API function stops the clock supply to the interval timer, by operating the RTCEN bit of peripheral enable register *n*.
For this reason, this API function also stops the clock supply to other peripheral devices sharing the RTCEN bit (e.g. real-timer clock).

[Classification]

r_cg_it.c

[Syntax]

```
void R_IT_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.11 Comparator

Below is a list of API functions output by the Code Generator for comparator use.

Table C-12. API Functions: [Comparator]

API Function Name	Function
R_COMP_Create	Performs initialization necessary to control the comparator functions.
R_COMP_Create_UserInit	Performs user-defined initialization relating to the comparator.
r_compn_interrupt	Performs processing in response to the comparator interrupt <i>INTCMPn</i> .
R_COMPn_Start	Begins comparison of reference input voltage and analog input voltage.
R_COMPn_Stop	Stops comparison of reference input voltage and analog input voltage.
R_COMP_Set_PowerOff	Halts the clock supplied to the comparator.

R_COMP_Create

Performs initialization necessary to control the comparator functions.

[Classification]

r_cg_comp.c

[Syntax]

```
void R_COMP_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_COMP_Create_UserInit

Performs user-defined initialization relating to the comparator.

Remark This API function is called as the [R_COMP_Create](#) callback routine.

[Classification]

r_cg_comp_user.c

[Syntax]

```
void R_COMP_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_comp n _interrupt

Performs processing in response to the comparator interrupt INTCMP n .

Remark This API function is called as the interrupt process corresponding to the comparator interrupt INTCMP n .

[Classification]

r_cg_comp_user.c

[Syntax]

```
__interrupt static void r_comp $n$ _interrupt ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP n _Start

Begins comparison of reference input voltage and analog input voltage.

[Classification]

r_cg_comp.c

[Syntax]

```
void R_COMP $n$ _Start ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP n _Stop

Stops comparison of reference input voltage and analog input voltage.

[Classification]

r_cg_comp.c

[Syntax]

```
void R_COMP $n$ _Stop ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP_Set_PowerOff

Halts the clock supplied to the comparator.

Remark Calling this API function changes the comparator to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_cg_comp.c

[Syntax]

```
void R_COMP_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.12 Clock Output/Buzzer Output

Below is a list of API functions output by the Code Generator for clock output/buzzer output use.

Table C-13. API Functions: [Clock Output/Buzzer Output]

API Function Name	Function
R_PCLBUZn_Create	Performs initialization necessary to control clock/buzzer output controller functions.
R_PCLBUZn_Create_UserInit	Performs user-defined initialization relating to the clock/buzzer output controller.
R_PCLBUZn_Start	Starts clock/buzzer output.
R_PCLBUZn_Stop	Ends clock/buzzer output.
R_PCLBUZ_Set_PowerOff	Halts the clock supplied to the clock/buzzer output controller.

R_PCLBUZn_Create

Performs initialization necessary to control clock/buzzer output controller functions.

[Classification]

r_cg_pclbuz.c

[Syntax]

```
void R_PCLBUZn_Create ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZn_Create_UserInit

Performs user-defined initialization relating to the clock/buzzer output controller.

Remark This API function is called as the [R_PCLBUZn_Create](#) callback routine.

[Classification]

r_cg_pclbuz_user.c

[Syntax]

```
void R_PCLBUZn_Create_UserInit ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ*n*_Start

Starts clock/buzzer output.

[Classification]

r_cg_pclbuz.c

[Syntax]

```
void R_PCLBUZn_Start ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ*n*_Stop

Ends clock/buzzer output.

[Classification]

r_cg_pclbuz.c

[Syntax]

```
void R_PCLBUZn_Stop ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ_Set_PowerOff

Halts the clock supplied to the clock/buzzer output controller.

- Remarks 1.** Calling this API function changes the clock/buzzer output controller to reset status. For this reason, writes to the control registers after this API function is called are ignored.
- 2.** This API function stops the clock supply to the clock/buzzer output controller, by operating the RTCEN bit of peripheral enable register *n*. For this reason, this API function also stops the clock supply to other peripheral devices sharing the RTCEN bit (e.g. real-time clock).

Remark**[Classification]**

r_cg_pclbuz.c

[Syntax]

```
void R_PCLBUZ_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.13 Data Transfer Controller

Below is a list of API functions output by the Code Generator for data transfer controller use.

Table C-14. API Functions: [Data Transfer Controller]

API Function Name	Function
R_DTC_Create	Performs initialization necessary to control the data transfer controller functions.
R_DTC_Create_UserInit	Performs user-defined initialization relating to the data transfer controller.
R_DTCn_Start	Enables operation of the data transfer controller.
R_DTCn_Stop	Disables operation of the data transfer controller.
R_DTC_Set_PowerOff	Halts the clock supplied to the data transfer controller.

R_DTC_Create

Performs initialization necessary to control the data transfer controller functions.

[Classification]

r_cg_dtc.c

[Syntax]

```
void R_DTC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTC_Create_UserInit

Performs user-defined initialization relating to the data transfer controller.

Remark This API function is called as the [R_DTC_Create](#) callback routine.

[Classification]

r_cg_dtc_user.c

[Syntax]

```
void R_DTC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTCn_Start

Enables operation of the data transfer controller.

[Classification]

r_cg_dtc.c

[Syntax]

```
void R_DTCn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTCn_Stop

Disables operation of the data transfer controller.

[Classification]

r_cg_dtc.c

[Syntax]

```
void R_DTCn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTC_Set_PowerOff

Halts the clock supplied to the data transfer controller.

Remark Calling this API function changes the data transfer controller to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_cg_dtc.c

[Syntax]

```
void R_DTC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.14 Event Link Controller

Below is a list of API functions output by the Code Generator for event link controller use.

Table C-15. API Functions: [Event Link Controller]

API Function Name	Function
R_ELC_Create	Performs initialization necessary to control the event link controller functions.
R_ELC_Create_UserInit	Performs user-defined initialization relating to the event link controller.
R_ELC_Stop	Disables operation of the event link controller.

R_ELC_Create

Performs initialization necessary to control the event link controller functions.

[Classification]

r_cg_elc.c

[Syntax]

```
void R_ELC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_Create_UserInit

Performs user-defined initialization relating to the event link controller.

Remark This API function is called as the [R_ELC_Create](#) callback routine.

[Classification]

r_cg_elc_user.c

[Syntax]

```
void R_ELC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_Stop

Disables operation of the event link controller.

[Classification]

r_cg_elc.c

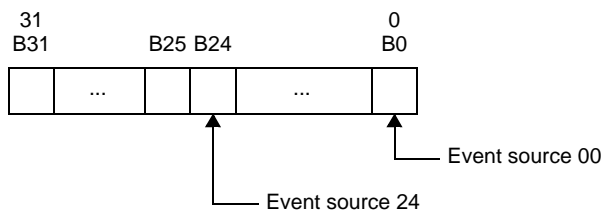
[Syntax]

```
void R_ELC_Stop ( uint32_t event );
```

[Argument(s)]

I/O	Argument	Description
I	uint32_t <i>event</i> ;	Disabled event source

Remark Below is shown the format for specifying disabled event source *event*.
 In case of setting the *event* to 0x01010101, the event link operations of event source 00, 08, 16, 24 are prohibited.



[Return value]

None.

C.3.15 LCD Controller/Driver

Below is a list of API functions output by the Code Generator for LCD controller/driver use.

Table C-16. API Functions: [LCD Controller/Driver]

API Function Name	Function
R_LCD_Create	Performs initialization necessary to control LCD controller/driver functions.
R_LCD_Create_UserInit	Performs user-defined initialization relating to the LCD controller/driver.
r_lcd_interrupt	Performs processing in response to the LCD frame interrupt INTLCD.
R_LCD_Start	Sets the LCD controller/driver to display on status.
R_LCD_Stop	Sets the LCD controller/driver to display off status.
R_LCD_Set_VoltageOn	Enables operation of internal voltage boost circuit and capacitor split circuit.
R_LCD_Set_VoltageOff	Disables operation of internal voltage boost circuit and capacitor split circuit.
R_LCD_Set_PowerOff	Halts the clock supplied to the LCD controller/driver.

R_LCD_Create

Performs initialization necessary to control LCD controller/driver functions.

[Classification]

r_cg_lcd.c

[Syntax]

```
void R_LCD_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Create_UserInit

Performs user-defined initialization relating to the LCD controller/driver.

Remark This API function is called as the [R_LCD_Create](#) callback routine.

[Classification]

r_cg_lcd_user.c

[Syntax]

```
void R_LCD_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lcd_interrupt

Performs processing in response to the LCD frame interrupt INTLCD.

Remark This API function is called as the interrupt process corresponding to the LCD frame interrupt INTLCD.

[Classification]

r_cg_lcd_user.c

[Syntax]

```
__interrupt static void r_lcd_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Start

Sets the LCD controller/driver to display on status.

[Classification]

r_cg_lcd.c

[Syntax]

```
void R_LCD_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Stop

Sets the LCD controller/driver to display off status.

[Classification]

r_cg_lcd.c

[Syntax]

```
void R_LCD_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Set_VoltageOn

Enables operation of internal voltage boost circuit and capacitor split circuit.

[Classification]

r_cg_lcd.c

[Syntax]

```
void R_LCD_Set_VoltageOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Set_VoltageOff

Disables operation of internal voltage boost circuit and capacitor split circuit.

[Classification]

r_cg_lcd.c

[Syntax]

```
void R_LCD_Set_VoltageOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Set_PowerOff

Halts the clock supplied to the LCD controller/driver.

- Remarks 1.** Calling this API function changes the LCD controller/driver to reset status. For this reason, writes to the control registers after this API function is called are ignored.
- 2.** This API function stops the clock supply to the LCD controller/driver, by operating the RTCEN bit of peripheral enable register *n*. For this reason, this API function also stops the clock supply to other peripheral devices sharing the RTCEN bit (e.g. real-time clock).

[Classification]

r_cg_lcd.c

[Syntax]

```
void R_LCD_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.16 Sound Generator

Below is a list of API functions output by the Code Generator for sound generator use.

Table C-17. API Functions: [Sound Generator]

API Function Name	Function
R_SG_Create	Performs initialization necessary to control sound generator functions.
R_SG_Create_UserInit	Performs user-defined initialization relating to the sound generator.
r_sg_interrupt	Performs processing in response to the threshold value detection of the logarithmic decrement interrupt INTSG.
R_SG_Start	Enables operation of sound generator.
R_SG_Stop	Disables operation of sound generator.

R_SG_Create

Performs initialization necessary to control sound generator functions.

[Classification]

r_cg_sg.c

[Syntax]

```
void R_SG_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SG_Create_UserInit

Performs user-defined initialization relating to the sound generator.

Remark This API function is called as the [R_SG_Create](#) callback routine.

[Classification]

r_cg_sg_user.c

[Syntax]

```
void R_SG_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_sg_interrupt

Performs processing in response to the threshold value detection of the logarithmic decrement interrupt INTSG.

Remark This API function is called as the interrupt process corresponding to the threshold value detection of the logarithmic decrement interrupt INTSG.

[Classification]

r_cg_sg_user.c

[Syntax]

```
__interrupt static void r_sg_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SG_Start

Enables operation of sound generator.

[Classification]

r_cg_sg.c

[Syntax]

```
void R_SG_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SG_Stop

Disables operation of sound generator.

[Classification]

r_cg_sg.c

[Syntax]

```
void R_SG_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.17 DMA Controller

Below is a list of API functions output by the Code Generator for DMA (Direct Memory Access) controller use.

Table C-18. API Functions: [DMA Controller]

API Function Name	Function
R_DMACn_Create	Performs initialization necessary to control DMA controller functions.
R_DMACn_Create_UserInit	Performs user-defined initialization relating to the DMA controller.
R_DMAC_Create	Performs initialization necessary to control DMA controller functions.
R_DMAC_Create_UserInit	Performs user-defined initialization relating to the DMA controller.
r_dmacn_interrupt	Performs processing in response to the DMA transfer end interrupt INTDMA n .
R_DMACn_Start	Enables operation of channel n .
R_DMACn_Stop	Disables operation of channel n .
R_DMACn_Set_SoftwareTriggerOn	Starts DMA transfer.

R_DMAc_n_Create

Performs initialization necessary to control DMA controller functions.

[Classification]

r_cg_dmac.c

[Syntax]

```
void R_DMAcn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAc n _Create_UserInit

Performs user-defined initialization relating to the DMA controller.

Remark This API function is called as the [R_DMAc \$n\$ _Create](#) callback routine.

[Classification]

r_cg_dmac_user.c

[Syntax]

```
void R_DMAc $n$ _Create_UserInit ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC_Create

Performs initialization necessary to control DMA controller functions.

[Classification]

r_cg_dmac.c

[Syntax]

```
void R_DMAC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DMAC_Create_UserInit

Performs user-defined initialization relating to the DMA controller.

Remark This API function is called as the [R_DMAC_Create](#) callback routine.

[Classification]

r_cg_dmac_user.c

[Syntax]

```
void R_DMAC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dmacn_interrupt

Performs processing in response to the DMA transfer end interrupt INTDMA n .

Remark This API function is called as the interrupt process corresponding to the DMA transfer end interrupt INTDMA n .

[Classification]

r_cg_dmac_user.c

[Syntax]

```
__interrupt static void r_dmacn_interrupt ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMACH_n_Start

Enables operation of channel *n*.

[Classification]

r_cg_dmac.c

[Syntax]

```
void R_DMACHn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAn_Stop

Disables operation of channel *n*.

- Remarks**
1. This API function does not forcibly terminate DMA transfer.
 2. Before using this API function, you must confirm that transmission has ended.

[Classification]

r_cg_dmac.c

[Syntax]

```
void R_DMAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC n _Set_SoftwareTriggerOn

Starts DMA transfer.

[Classification]

r_cg_dmac.c

[Syntax]

```
void R_DMAC $n$ _Set_SoftwareTriggerOn ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.3.18 Voltage Detector

Below is a list of API functions output by the Code Generator for voltage detector use.

Table C-19. API Functions: [Voltage Detector]

API Function Name	Function
R_LVD_Create	Performs initialization necessary to control voltage detector functions.
R_LVD_Create_UserInit	Performs user-defined initialization relating to the voltage detector.
r_lvd_interrupt	Performs processing in response to the voltage detection interrupt INTLVI.
R_LVD_InterruptMode_Start	Starts voltage detection (when in interrupt mode, and interrupt & reset mode).

R_LVD_Create

Performs initialization necessary to control voltage detector functions.

[Classification]

r_cg_lvd.c

[Syntax]

```
void R_LVD_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Create_UserInit

Performs user-defined initialization relating to the voltage detector.

Remark This API function is called as the [R_LVD_Create](#) callback routine.

[Classification]

r_cg_lvd_user.c

[Syntax]

```
void R_LVD_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lvd_interrupt

Performs processing in response to the voltage detection interrupt INTLVI.

Remark This API function is called as the interrupt process corresponding to the voltage detection interrupt INTLVI.

[Classification]

r_cg_lvd_user.c

[Syntax]

```
__interrupt static void r_lvd_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_InterruptMode_Start

Starts voltage detection (when in interrupt mode, and interrupt & reset mode).

[Classification]

r_cg_lvd.c

[Syntax]

```
void R_LVD_InterruptMode_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.19 Programmable Gain Amplifier

Below is a list of API functions output by the Code Generator for programmable gain amplifiers use.

Table C-20. API Functions: [Programmable Grain Amplifier]

API Function Name	Function
R_PGA_Create	Performs initialization necessary to control programmable gain amplifiers functions.
R_PGA_Create_UserInit	Performs user-defined initialization relating to the programmable gain amplifiers.
R_PGA_Start	Starts the operation of programmable gain amplifier.
R_PGA_Stop	Ends the operation of programmable gain amplifier.

R_PGA_Create

Performs initialization necessary to control programmable gain amplifiers functions.

[Classification]

r_cg_pga.c

[Syntax]

```
void R_PGA_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Create_UserInit

Performs user-defined initialization relating to the programmable gain amplifiers.

Remark This API function is called as the [R_PGA_Create](#) callback routine.

[Classification]

r_cg_pga_user.c

[Syntax]

```
void R_PGA_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Start

Starts the operation of programmable gain amplifier.

[Classification]

r_cg_pga.c

[Syntax]

```
void R_PGA_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Stop

Ends the operation of programmable gain amplifier.

[Classification]

r_cg_pga.c

[Syntax]

```
void R_PGA_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

APPENDIX D INDEX

A

A/D Converter ... 214

- R_ADC_Create ... 215
- R_ADC_Create_UserInit ... 216
- R_ADC_Get_Result ... 227
- R_ADC_Get_Result_8bit ... 228
- r_adc_interrupt ... 217
- R_ADC_Set_ADChannel ... 223
- R_ADC_Set_OperationOff ... 219
- R_ADC_Set_OperationOn ... 218
- R_ADC_Set_PowerOff ... 222
- R_ADC_Set_SnoozeOff ... 225
- R_ADC_Set_SnoozeOn ... 224
- R_ADC_Set_TestChannel ... 226
- R_ADC_Start ... 220
- R_ADC_Stop ... 221

API functions ... 91

- A/D Converter ... 214
- Clock Generator ... 103
- Clock Output/Buzzer Output ... 334
- Comparator ... 327
- D/A Converter ... 229
- Data Transfer Controller ... 340
- DMA Controller ... 365
- Event Link Controller ... 346
- Interrupt ... 113
- Interval Timer ... 320
- LCD Controller/Driver ... 350
- Port ... 110
- Programmable Gain Amplifier ... 379
- Real-time Clock ... 299
- Serial ... 127
- Sound Generator ... 359
- Timer ... 236
- Voltage Detector ... 374
- Watchdog Timer ... 294

B

Browse For Folder dialog box ... 81

C

Clock Generator ... 103

- r_cgc_clockmonitor_interrupt ... 106
- R_CGC_Create ... 104
- R_CGC_Create_UserInit ... 105
- R_CGC_Get_ResetSource ... 107
- R_CGC_Set_ClockMode ... 108
- R_CGC_Set_CRCon ... 109

Clock Output/Buzzer Output ... 334

- R_PCLBUZn_Create ... 335
- R_PCLBUZn_Create_UserInit ... 336
- R_PCLBUZn_Start ... 337
- R_PCLBUZn_Stop ... 338
- R_PCLBUZ_Set_PowerOff ... 339

Code Generator panel ... 69

Code Generator Preview panel ... 72

Column Chooser dialog box ... 77

Comparator ... 327

- R_COMP_Create ... 328
- R_COMP_Create_UserInit ... 329
- r_compn_interrupt ... 330
- R_COMPn_Start ... 331
- R_COMPn_Stop ... 332
- R_COMP_Set_PowerOff ... 333

D

D/A Converter ... 229

- R_DAC_Create ... 230
- R_DAC_Create_UserInit ... 231
- R_DACn_Set_ConversionValue ... 235
- R_DACn_Start ... 232
- R_DACn_Stop ... 233
- R_DAC_Set_PowerOff ... 234

Data Transfer Controller ... 340

- R_DTC_Create ... 341

- R_DTC_Create_UserInit ... 342
- R_DTCn_Start ... 343
- R_DTCn_Stop ... 344
- R_DTC_Set_PowerOff ... 345
- [Device Pin List Information] tab ... 49
- Device Pin List panel ... 58
 - [External Peripheral] tab ... 64
 - [Macro] tab ... 62
 - [Pin Number] tab ... 60
- Device Top View panel ... 66
- [Device Top View Settings] tab ... 50
- DMA Controller ... 365
 - R_DMAC_Create ... 368
 - R_DMAC_Create_UserInit ... 369
 - R_DMACn_Create ... 366
 - R_DMACn_Create_UserInit ... 367
 - r_dmacn_interrupt ... 370
 - R_DMACn_Set_SoftwareTriggerOn ... 373
 - R_DMACn_Start ... 371
 - R_DMACn_Stop ... 372
- E**
- Event Link Controller ... 346
 - R_ELC_Create ... 347
 - R_ELC_Create_UserInit ... 348
 - R_ELC_Stop ... 349
- [External Peripheral] tab ... 64
- F**
- [File Setting] tab ... 57
- Functions ... 8, 22
 - Code Generator ... 22
 - Pin Configurator ... 8
- G**
- [Generation] tab ... 53
- I**
- Interrupt ... 113
 - R_INTC_Create ... 114
 - R_INTC_Create_UserInit ... 115
 - r_intclrn_interrupt ... 124
- R_INTCLRn_Start ... 125
- R_INTCLRn_Stop ... 126
- r_intcn_interrupt ... 116
- R_INTCn_Start ... 117
- R_INTCn_Stop ... 118
- R_KEY_Create ... 119
- R_KEY_Create_UserInit ... 120
- r_key_interrupt ... 121
- R_KEY_Start ... 122
- R_KEY_Stop ... 123
- Interval Timer ... 320
 - R_IT_Create ... 321
 - R_IT_Create_UserInit ... 322
 - r_it_interrupt ... 323
 - R_IT_Set_PowerOff ... 326
 - R_IT_Start ... 324
 - R_IT_Stop ... 325
- L**
- LCD Controller/Driver ... 350
 - R_LCD_Create ... 351
 - R_LCD_Create_UserInit ... 352
 - r_lcd_interrupt ... 353
 - R_LCD_Set_PowerOff ... 358
 - R_LCD_Set_VoltageOff ... 357
 - R_LCD_Set_VoltageOn ... 356
 - R_LCD_Start ... 354
 - R_LCD_Stop ... 355
- M**
- [Macro Setting] tab ... 56
- [Macro] tab ... 62
- Main window ... 39
- N**
- New Column dialog box ... 80
- O**
- Output panel ... 75
- P**
- [Pin Configurator Settings] tab ... 48

- [Pin Number] tab ... 60
- Port ... 110
 - R_PORT_Create ... 111
 - R_PORT_Create_UserInit ... 112
- Programmable Gain Amplifier ... 379
 - R_PGA_Create ... 380
 - R_PGA_Create_UserInit ... 381
 - R_PGA_Start ... 382
 - R_PGA_Stop ... 383
- Project Tree panel ... 42
- Property panel ... 45
 - [Device Pin List Information] tab ... 49
 - [Device Top View Settings] tab ... 50
 - [File Setting] tab ... 57
 - [Generation] tab ... 53
 - [Macro Setting] tab ... 56
 - [Pin Configurator Settings] tab ... 48
- R**
- R_ADC_Create ... 215
- R_ADC_Create_UserInit ... 216
- R_ADC_Get_Result ... 227
- R_ADC_Get_Result_8bit ... 228
- r_adc_interrupt ... 217
- R_ADC_Set_ADChannel ... 223
- R_ADC_Set_OperationOff ... 219
- R_ADC_Set_OperationOn ... 218
- R_ADC_Set_PowerOff ... 222
- R_ADC_Set_SnoozeOff ... 225
- R_ADC_Set_SnoozeOn ... 224
- R_ADC_Set_TestChannel ... 226
- R_ADC_Start ... 220
- R_ADC_Stop ... 221
- r_cgc_clockmonitor_interrupt ... 106
- R_CGC_Create ... 104
- R_CGC_Create_UserInit ... 105
- R_CGC_Get_ResetSource ... 107
- R_CGC_Set_ClockMode ... 108
- R_CGC_Set_CRCon ... 109
- R_COMP_Create ... 328
- R_COMP_Create_UserInit ... 329
- r_compn_interrupt ... 330
- R_COMPn_Start ... 331
- R_COMPn_Stop ... 332
- R_COMP_Set_PowerOff ... 333
- r_csimn_callback_error ... 156
- r_csimn_callback_receiveend ... 155
- r_csimn_callback_sendend ... 154
- R_CSImn_Create ... 147
- r_csimn_interrupt ... 148
- R_CSImn_Receive ... 152
- R_CSImn_Send ... 151
- R_CSImn_Send_Receive ... 153
- R_CSImn_Start ... 149
- R_CSImn_Stop ... 150
- R_DAC_Create ... 230
- R_DAC_Create_UserInit ... 231
- R_DACn_Set_ConversionValue ... 235
- R_DACn_Start ... 232
- R_DACn_Stop ... 233
- R_DAC_Set_PowerOff ... 234
- r_dalin_callback_error ... 177
- r_dalin_callback_receiveend ... 176
- r_dalin_callback_sendend ... 175
- r_dalin_callback_softwareoverrun ... 178
- R_DALIn_Create ... 167
- r_dalin_interrupt_error ... 170
- r_dalin_interrupt_receive ... 169
- r_dalin_interrupt_send ... 168
- R_DALIn_Receive ... 174
- R_DALIn_Send ... 173
- R_DALIn_Start ... 171
- R_DALIn_Stop ... 172
- R_DMxAC_Create ... 368
- R_DMxAC_Create_UserInit ... 369
- R_DMxACn_Create ... 366
- R_DMxACn_Create_UserInit ... 367
- r_dmacn_interrupt ... 370
- R_DMxACn_Set_SoftwareTriggerOn ... 373
- R_DMxACn_Start ... 371
- R_DMxACn_Stop ... 372
- R_DTC_Create ... 341

R_DTC_Create_UserInit ...	342	R_IICAn_Slave_Send ...	208
R_DTCn_Start ...	343	R_IICAn_Stop ...	201
R_DTCn_Stop ...	344	R_IICAn_StopCondition ...	200
R_DTC_Set_PowerOff ...	345	r_iicmn_callback_master_error ...	166
Real-time Clock ...	299	r_iicmn_callback_master_receiveend ...	165
r_rtc_callback_alarm ...	317	r_iicmn_callback_master_sendend ...	164
r_rtc_callback_constperiod ...	311	R_IICmn_Create ...	157
R_RTC_Create ...	300	r_iicmn_interrupt ...	158
R_RTC_Create_UserInit ...	301	R_IICmn_Master_Receive ...	163
R_RTC_Get_AlarmValue ...	316	R_IICmn_Master_Send ...	162
R_RTC_Get_CounterValue ...	308	R_IICmn_StartCondition ...	159
r_rtc_interrupt ...	302	R_IICmn_Stop ...	161
R_RTC_Set_AlarmOff ...	313	R_IICmn_StopCondition ...	160
R_RTC_Set_AlarmOn ...	312	R_INTC_Create ...	114
R_RTC_Set_AlarmValue ...	314	R_INTC_Create_UserInit ...	115
R_RTC_Set_CounterValue ...	307	r_intclrn_interrupt ...	124
R_RTC_Set_HourSystem ...	306	R_INTCLRn_Start ...	125
R_RTC_Set_PowerOff ...	305	R_INTCLRn_Stop ...	126
R_RTC_Set_RTC1HZOff ...	319	r_intcn_interrupt ...	116
R_RTC_Set_RTC1HZOn ...	318	R_INTCn_Start ...	117
R_RTC_Start ...	303	R_INTCn_Stop ...	118
R_RTC_Stop ...	304	R_IT_Create ...	321
R_RTC_Set_ConstPeriodInterruptOff ...	310	R_LCD_Create ...	351
R_RTC_Set_ConstPeriodInterruptOn ...	309	R_IT_Create_UserInit ...	322
R_ELC_Create ...	347	r_it_interrupt ...	323
R_ELC_Create_UserInit ...	348	R_IT_Set_PowerOff ...	326
R_ELC_Stop ...	349	R_IT_Start ...	324
r_iican_callback_getstopcondition ...	213	R_IT_Stop ...	325
r_iican_callback_master_error ...	207	R_KEY_Create ...	119
r_iican_callback_master_receiveend ...	206	R_KEY_Create_UserInit ...	120
r_iican_callback_master_sendend ...	205	r_key_interrupt ...	121
r_iican_callback_slave_error ...	212	R_KEY_Start ...	122
r_iican_callback_slave_receiveend ...	211	R_KEY_Stop ...	123
r_iican_callback_slave_sendend ...	210	R_LCD_Create_UserInit ...	352
R_IICAn_Create ...	197	r_lcd_interrupt ...	353
R_IICAn_Create_UserInit ...	198	R_LCD_Set_PowerOff ...	358
r_iican_interrupt ...	199	R_LCD_Set_VoltageOff ...	357
R_IICAn_Master_Receive ...	204	R_LCD_Set_VoltageOn ...	356
R_IICAn_Master_Send ...	203	R_LCD_Start ...	354
R_IICAn_Set_PowerOff ...	202	R_LCD_Stop ...	355
R_IICAn_Slave_Receive ...	209	R_LVD_Create ...	375

R_LVD_Create_UserInit ...	376	R_SG_Start ...	363
r_lvd_interrupt ...	377	R_SG_Stop ...	364
R_LVD_InterruptMode_Start ...	378	R_TAUm_Channeln_Get_PulseWidth ...	249
R_PCLBUZn_Create ...	335	r_taum_channeln_higher8bits_interrupt ...	241
R_PCLBUZn_Create_UserInit ...	336	R_TAUm_Channeln_Higher8bits_Start ...	243
R_PCLBUZn_Start ...	337	R_TAUm_Channeln_Higher8bits_Stop ...	246
R_PCLBUZn_Stop ...	338	r_taum_channeln_interrupt ...	240
R_PCLBUZ_Set_PowerOff ...	339	R_TAUm_Channeln_Lower8bits_Start ...	244
R_PGA_Create ...	380	R_TAUm_Channeln_Lower8bits_Stop ...	247
R_PGA_Create_UserInit ...	381	R_TAUm_Channeln_Set_SoftwareTriggerOn ...	250
R_PGA_Start ...	382	R_TAUm_Channeln_Start ...	242
R_PGA_Stop ...	383	R_TAUm_Channeln_Stop ...	245
R_PORT_Create ...	111	R_TAUm_Create ...	238
R_PORT_Create_UserInit ...	112	R_TAUm_Create_UserInit ...	239
r_rtc_callback_alarm ...	317	R_TAUm_Set_PowerOff ...	248
r_rtc_callback_constperiod ...	311	R_TMR_KB_Create ...	274
R_RTC_Create ...	300	R_TMR_KBm_Create_UserInit ...	275
R_RTC_Create_UserInit ...	301	R_TMR_KBm_ForcedOutput_Start ...	280
R_RTC_Get_AlarmValue ...	316	R_TMR_KBm_ForcedOutput_Stop ...	281
R_RTC_Get_CounterValue ...	308	r_tmr_kbm_interrupt ...	276
r_rtc_interrupt ...	302	R_TMR_KBm_Set_PowerOff ...	279
R_RTC_Set_AlarmOff ...	313	R_TMR_KBm_Start ...	277
R_RTC_Set_AlarmOn ...	312	R_TMR_KBm_Stop ...	278
R_RTC_Set_AlarmValue ...	314	R_TMR_KC0_Create ...	282
R_RTC_Set_CounterValue ...	307	R_TMR_KC0_Create_UserInit ...	283
R_RTC_Set_HourSystem ...	306	r_tmr_kc0_interrupt ...	284
R_RTC_Set_PowerOff ...	305	R_TMR_KC0_Set_PowerOff ...	287
R_RTC_Set_RTC1HZOff ...	319	R_TMR_KC0_Start ...	285
R_RTC_Set_RTC1HZOn ...	318	R_TMR_KC0_Stop ...	286
R_RTC_Start ...	303	R_TMR_RDn_Create ...	251
R_RTC_Stop ...	304	R_TMR_RDn_Create_UserInit ...	252
R_RTC_Set_ConstPeriodInterruptOff ...	310	R_TMR_RDn_ForcedOutput_Start ...	257
R_RTC_Set_ConstPeriodInterruptOn ...	309	R_TMR_RDn_ForcedOutput_Stop ...	258
R_SAUm_Create ...	130	R_TMR_RDn_Get_PulseWidth ...	259
R_SAUm_Create_UserInit ...	131	r_tmr_rdn_interrupt ...	253
R_SAUm_Set_PowerOff ...	132	R_TMR_RDn_Set_PowerOff ...	256
R_SAUm_Set_SnoozeOff ...	134	R_TMR_RDn_Start ...	254
R_SAUm_Set_SnoozeOn ...	133	R_TMR_RDn_Stop ...	255
R_SG_Create ...	360	R_TMR_RG0_Create ...	260
R_SG_Create_UserInit ...	361	R_TMR_RG0_Create_UserInit ...	261
r_sg_interrupt ...	362	R_TMR_RG0_Get_PulseWidth ...	266

- r_tmr_rg0_interrupt ... 262
 - R_TMR_RG0_Set_PowerOff ... 265
 - R_TMR_RG0_Start ... 263
 - R_TMR_RG0_Stop ... 264
 - R_TMR_RJ0_Create ... 267
 - R_TMR_RJ0_Create_UserInit ... 268
 - R_TMR_RJ0_Get_PulseWidth ... 273
 - r_tmr_rj0_interrupt ... 269
 - R_TMR_RJ0_Set_PowerOff ... 272
 - R_TMR_RJ0_Start ... 270
 - R_TMR_RJ0_Stop ... 271
 - r_uartfn_callback_error ... 193
 - r_uartfn_callback_expbitdetect ... 195
 - r_uartfn_callback_idmatch ... 196
 - r_uartfn_callback_receiveend ... 192
 - r_uartfn_callback_sendend ... 191
 - r_uartfn_callback_softwareoverrun ... 194
 - R_UARTFn_Create ... 179
 - R_UARTFn_Create_UserInit ... 180
 - r_uartfn_interrupt_error ... 183
 - r_uartfn_interrupt_receive ... 182
 - r_uartfn_interrupt_send ... 181
 - R_UARTFn_Receive ... 188
 - R_UARTFn_Send ... 187
 - R_UARTFn_Set_DataComparisonOff ... 190
 - R_UARTFn_Set_DataComparisonOn ... 189
 - R_UARTFn_Set_PowerOff ... 186
 - R_UARTFn_Start ... 184
 - R_UARTFn_Stop ... 185
 - r_uartn_callback_error ... 145
 - r_uartn_callback_receiveend ... 144
 - r_uartn_callback_sendend ... 143
 - r_uartn_callback_softwareoverrun ... 146
 - R_UARTn_Create ... 135
 - r_uartn_interrupt_error ... 138
 - r_uartn_interrupt_receive ... 137
 - r_uartn_interrupt_send ... 136
 - R_UARTn_Receive ... 142
 - R_UARTn_Send ... 141
 - R_UARTn_Start ... 139
 - R_UARTn_Stop ... 140
 - R_WDT_Create ... 295
 - R_WDT_Create_UserInit ... 296
 - r_wdt_interrupt ... 297
 - R_WDT_Restart ... 298
 - R_WUTM_Create ... 288
 - R_WUTM_Create_UserInit ... 289
 - r_wuttm_interrupt ... 290
 - R_WUTM_Set_PowerOff ... 293
 - R_WUTM_Start ... 291
 - R_WUTM_Stop ... 292
- S**
- Save As dialog box ... 82
 - Serial ... 127
 - r_csimn_callback_error ... 156
 - r_csimn_callback_receiveend ... 155
 - R_CSImn_Create ... 147
 - r_csimn_interrupt ... 148
 - R_CSImn_Receive ... 152
 - R_CSImn_Send ... 151
 - r_csimn_callback_sendend ... 154
 - R_CSImn_Send_Receive ... 153
 - R_CSImn_Start ... 149
 - R_CSImn_Stop ... 150
 - r_dalin_callback_error ... 177
 - r_dalin_callback_receiveend ... 176
 - r_dalin_callback_sendend ... 175
 - r_dalin_callback_softwareoverrun ... 178
 - R_DALIn_Create ... 167
 - r_dalin_interrupt_error ... 170
 - r_dalin_interrupt_receive ... 169
 - r_dalin_interrupt_send ... 168
 - R_DALIn_Receive ... 174
 - R_DALIn_Send ... 173
 - R_DALIn_Start ... 171
 - R_DALIn_Stop ... 172
 - r_iican_callback_getstopcondition ... 213
 - r_iican_callback_master_error ... 207
 - r_iican_callback_master_receiveend ... 206
 - r_iican_callback_master_sendend ... 205
 - r_iican_callback_slave_error ... 212

- r_iican_callback_slave_receiveend ... 211
 - r_iican_callback_slave_sendend ... 210
 - R_IICAn_Create ... 197
 - R_IICAn_Create_UserInit ... 198
 - r_iican_interrupt ... 199
 - R_IICAn_Master_Receive ... 204
 - R_IICAn_Master_Send ... 203
 - R_IICAn_Set_PowerOff ... 202
 - R_IICAn_Slave_Receive ... 209
 - R_IICAn_Slave_Send ... 208
 - R_IICAn_Stop ... 201
 - R_IICAn_StopCondition ... 200
 - r_iicmn_callback_master_error ... 166
 - r_iicmn_callback_master_receiveend ... 165
 - r_iicmn_callback_master_sendend ... 164
 - R_IICmn_Create ... 157
 - r_iicmn_interrupt ... 158
 - R_IICmn_Master_Receive ... 163
 - R_IICmn_Master_Send ... 162
 - R_IICmn_StartCondition ... 159
 - R_IICmn_Stop ... 161
 - R_IICmn_StopCondition ... 160
 - R_SAUm_Create ... 130
 - R_SAUm_Create_UserInit ... 131
 - R_SAUm_Set_PowerOff ... 132
 - R_SAUm_Set_SnoozeOff ... 134
 - R_SAUm_Set_SnoozeOn ... 133
 - r_uartfn_callback_error ... 193
 - r_uartfn_callback_expbitdetect ... 195
 - r_uartfn_callback_idmatch ... 196
 - r_uartfn_callback_receiveend ... 192
 - r_uartfn_callback_sendend ... 191
 - r_uartfn_callback_softwareoverrun ... 194
 - R_UARTFn_Create ... 179
 - R_UARTFn_CreateUserInit ... 180
 - r_uartfn_interrupterror ... 183
 - r_uartfn_interruptreceive ... 182
 - r_uartfn_interruptsend ... 181
 - R_UARTFn_Receive ... 188
 - R_UARTFn_Send ... 187
 - R_UARTFn_Set_DataComparisonOff ... 190
 - R_UARTFn_Set_DataComparisonOn ... 189
 - R_UARTFn_Set_PowerOff ... 186
 - R_UARTFn_Start ... 184
 - R_UARTFn_Stop ... 185
 - r_uartn_callback_error ... 145
 - r_uartn_callback_receiveend ... 144
 - r_uartn_callback_sendend ... 143
 - r_uartn_callback_softwareoverrun ... 146
 - R_UARTn_Create ... 135
 - r_uartn_interrupt_error ... 138
 - r_uartn_interrupt_receive ... 137
 - r_uartn_interrupt_send ... 136
 - R_UARTn_Receive ... 142
 - R_UARTn_Send ... 141
 - R_UARTn_Start ... 139
 - R_UARTn_Stop ... 140
 - Sound Generator ... 359
 - R_SG_Create ... 360
 - R_SG_Create_UserInit ... 361
 - r_sg_interrupt ... 362
 - R_SG_Start ... 363
 - R_SG_Stop ... 364
- T**
- Timer ... 236
 - R_TAUm_Channeln_Get_PulseWidth ... 249
 - r_taum_channeln_higher8bits_interrupt ... 241
 - R_TAUm_Channeln_Higher8bits_Stop ... 246
 - r_taum_channeln_interrupt ... 240
 - R_TAUm_Channeln_Lower8bits_Start ... 244
 - R_TAUm_Channeln_Lower8bits_Stop ... 247
 - R_TAUm_Channeln_Set_SoftwareTriggerOn ... 250
 - R_TAUm_Channeln_Start ... 242
 - R_TAUm_Channeln_Stop ... 245
 - R_TAUm_Create ... 238
 - R_TAUm_Create_UserInit ... 239
 - R_TAUm_Set_PowerOff ... 248
 - R_TMR_KB_Create ... 274
 - R_TMR_KBm_Create_UserInit ... 275
 - R_TMR_KBm_ForcedOutput_Start ... 280

R_TMR_KBm_ForcedOutput_Stop ... 281
 r_tmr_kbm_interrupt ... 276
 R_TMR_KBm_Set_PowerOff ... 279
 R_TMR_KBm_Start ... 277
 R_TMR_KBm_Stop ... 278
 R_TMR_KC0_Create ... 282
 R_TMR_KC0_Create_UserInit ... 283
 r_tmr_kc0_interrupt ... 284
 R_TMR_KC0_Set_PowerOff ... 287
 R_TMR_KC0_Start ... 285
 R_TMR_KC0_Stop ... 286
 R_TMR_RDn_Create ... 251
 R_TMR_RDn_Create_UserInit ... 252
 R_TMR_RDn_ForcedOutput_Start ... 257
 R_TMR_RDn_ForcedOutput_Stop ... 258
 R_TMR_RDn_Get_PulseWidth ... 259
 r_tmr_rdn_interrupt ... 253
 R_TMR_RDn_Set_PowerOff ... 256
 R_TMR_RDn_Start ... 254
 R_TMR_RDn_Stop ... 255
 R_TMR_RG0_Create ... 260
 R_TMR_RG0_Create_UserInit ... 261
 R_TMR_RG0_Get_PulseWidth ... 266
 r_tmr_rg0_interrupt ... 262
 R_TMR_RG0_Set_PowerOff ... 265
 R_TMR_RG0_Start ... 263
 R_TMR_RG0_Stop ... 264
 R_TMR_RJ0_Create ... 267
 R_TMR_RJ0_Create_UserInit ... 268
 R_TMR_RJ0_Get_PulseWidth ... 273
 r_tmr_rj0_interrupt ... 269
 R_TMR_RJ0_Set_PowerOff ... 272
 R_TMR_RJ0_Start ... 270
 R_TMR_RJ0_Stop ... 271
 R_WUTM_Create ... 288
 R_WUTM_Create_UserInit ... 289
 r_wutm_interrupt ... 290
 R_WUTM_Set_PowerOff ... 293
 R_WUTM_Start ... 291
 R_WUTM_Stop ... 292
 R_TAUm_Channeln_Higher8bits_Start ... 243

V

Voltage Detector ... 374
 R_LVD_Create ... 375
 R_LVD_Create_UserInit ... 376
 r_lvd_interrupt ... 377
 R_LVD_InterruptMode_Start ... 378

W

Watchdog Timer ... 294
 R_WDT_Create ... 295
 R_WDT_Create_UserInit ... 296
 r_wdt_interrupt ... 297
 R_WDT_Restart ... 298
 Window reference ... 38

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep 01, 2012	-	First Edition issued

CubeSuite+ V1.03.00 User's Manual:
RL78 Design

Publication Date: Rev.1.00 Sep 01, 2012

Published by: Renesas Electronics Corporation

**SALES OFFICES**

Renesas Electronics Corporation

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

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852-2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, 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 906, 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 Korea Co., Ltd.
11F., Samik Lavied'or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

CubeSuite+ V1.03.00