User's Manual

CubeSuite+ V2.01.00

Integrated Development Environment

User's Manual: RX Design

Target Device
RX Family

Renesas Electronics
www.renesas.com

Rev.1.00 Sep 2013

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.

(2012.4)

How to Use This Manual

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

CubeSuite+ is an integrated development environment (IDE) for RX 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
APPENDIX A WINDOW REFERENCE
APPENDIX B OUTPUT FILES
APPENDIX C API FUNCTIONS

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:	Higher 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	R20UT2682E
	RX Design	This manual
	V850 Design	R20UT2134E
	R8C Design	R20UT2135E
	RL78 Design	R20UT2684E
	78K0R Design	R20UT2137E
	78K0 Design	R20UT2138E
	RH850 Coding	R20UT2584E
	RX Coding	R20UT2470E
	V850 Coding	R20UT0553E
	Coding for CX Compiler	R20UT2659E
	R8C Coding	R20UT0576E
	RL78, 78K0R Coding	R20UT2140E
	78K0 Coding	R20UT2141E
	RH850 Build	R20UT2585E
	RX Build	R20UT2472E
	V850 Build	R20UT0557E
	Build for CX Compiler	R20UT2142E
	R8C Build	R20UT0575E
	RL78, 78K0R Build	R20UT2143E
	78K0 Build	R20UT0783E
	RH850 Debug	R20UT2685E
	RX Debug	R20UT2702E
	V850 Debug	R20UT2446E
	R8C Debug	R20UT0770E
	RL78 Debug	R20UT2445E
	78K0R Debug	R20UT0732E
	78K0 Debug	R20UT0731E
	Analysis	R20UT2686E
	Message	R20UT2687E

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

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 (Code Generator).

1.1 Overview

The design tool, which is one of the components provided by CubeSuite+, enables you to output the source code (device driver programs, C source files and header files) necessary to control the peripheral functions (clock generation circuit, voltage detection circuit, etc.) provided by the device by configuring various information using the GUI.

1.2 Features

The design tool (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.

- Reporting function
  You can output configured information using the 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

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

Remark In this chapter, an example where an RX111 R5F51111AxFK(64pin) is the target device is used to explain the key functions.

2.1 Overview

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

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, device 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 Peripheral Functions Panel

Open the Peripheral Functions panel used to configure the information necessary to control the peripheral functions (clock generation circuit, voltage detection circuit, etc.).

(4) Enter Information

Configure the information necessary to control the peripheral functions (clock generation circuit, voltage detection circuit, etc.) in the Peripheral Functions panel.

(5) Confirm Source Code

Confirm the source code (device driver program) that reflects the information configured in the Peripheral Functions 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.

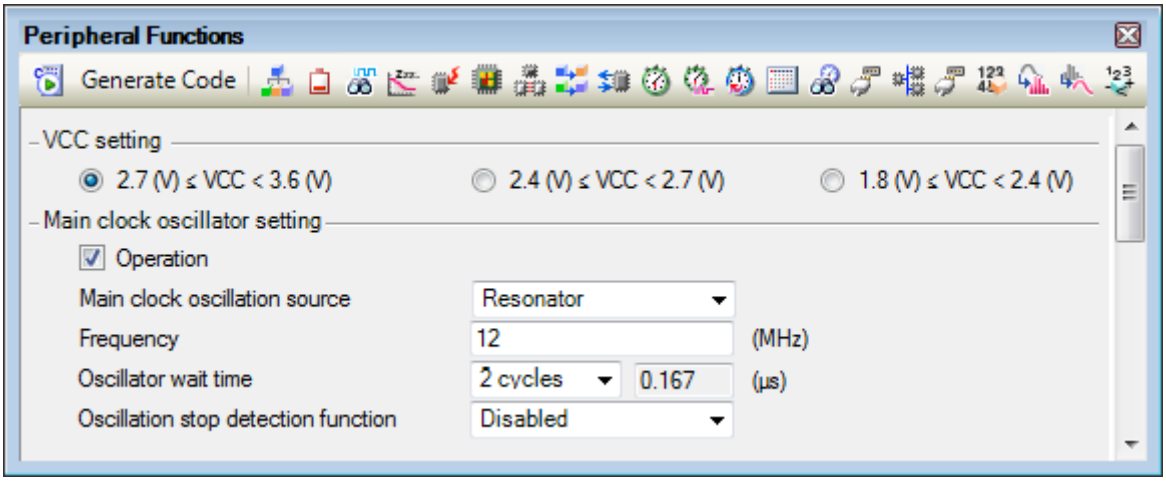
Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Save project".

2.2 Open Peripheral Functions Panel

The Peripheral Functions panel is opened to set the information necessary to control the peripheral functions (clock generation circuit, voltage detection circuit, etc.) provided in the device.

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

Figure 2-1. Open Peripheral Functions Panel



Remark If an unsupported device 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.

2.3 Enter Information

Configure the information necessary to control the peripheral functions (clock generation circuit, voltage detection circuit, etc.) in the information setting area of the Peripheral Functions panel which is opened as described in "2.2 Open Peripheral Functions Panel".

Remark When controlling multiple peripheral functions, repeat the procedures described in "2.2 Open Peripheral Functions Panel" through "2.3 Enter Information".

2.3.1 Input rule

Following is the rules for input to the Peripheral Functions panel.

(1) Character set

Character sets that are allowed to input are as follows.

Table 2-1. 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 2-2. List of Notation

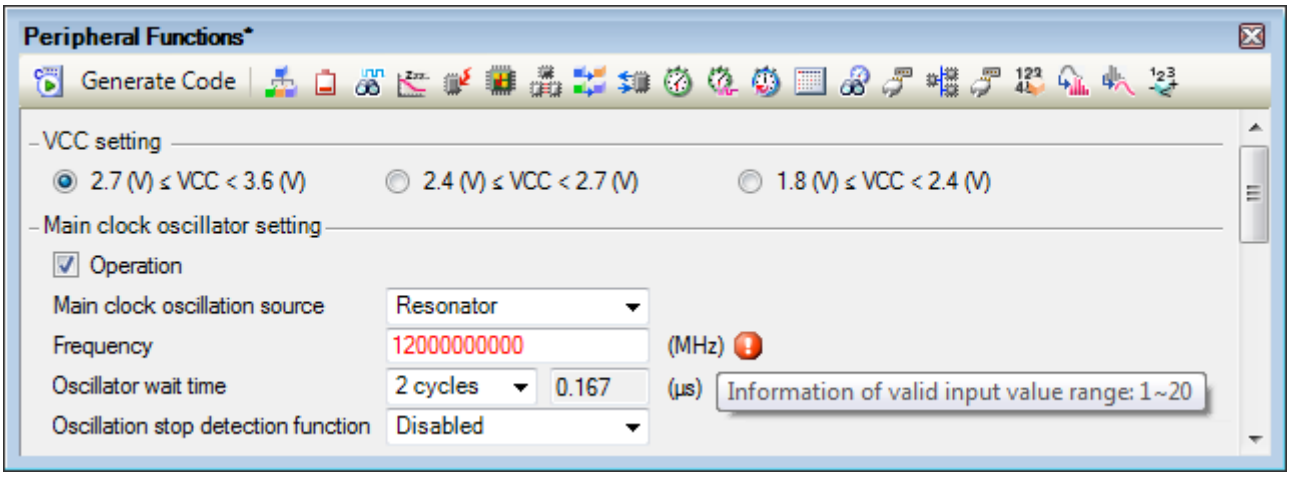
Notation	Outline
Decimal number	A numeric value that starts with a number between 1 and 9 and followed by numbers between 0 and 9, and the numeric value 0
Hex number	A numeric value that starts with 0x and followed by a combination of numbers from 0 to 9 and characters from A to F (characters are not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

2.3.2 Icon indicating incorrect entry

When performing code generation, if you enter an invalid string in the Peripheral Functions 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) popups.

Figure 2-2. Icon Indicating Incorrect Entry

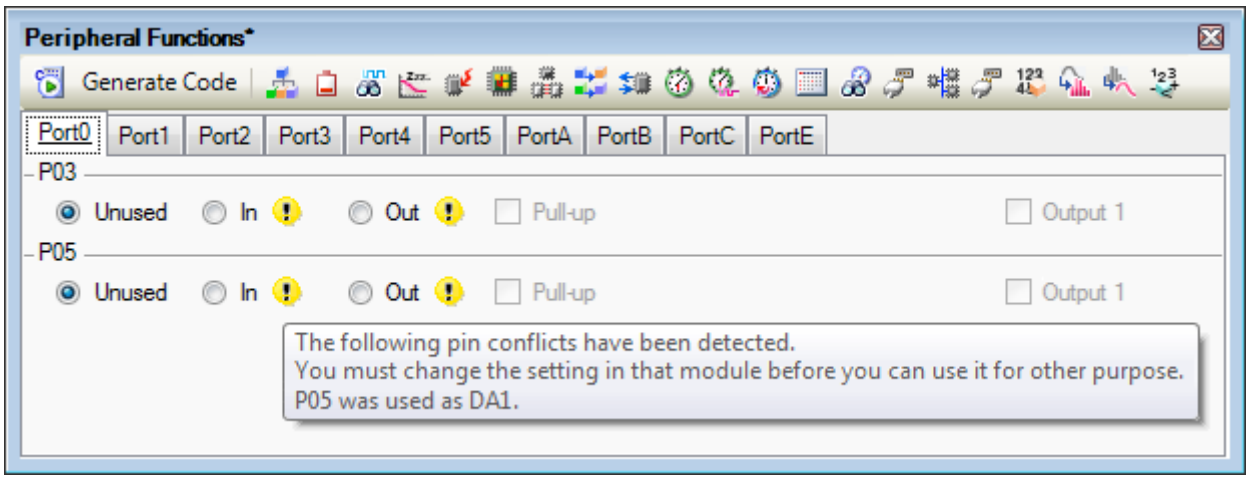


2.3.3 Icon indicating pin conflict

If a conflict occurs between the pins while setting various peripheral functions in the Peripheral Functions 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) popups.

Figure 2-3. Icon Indicating Pin Conflict

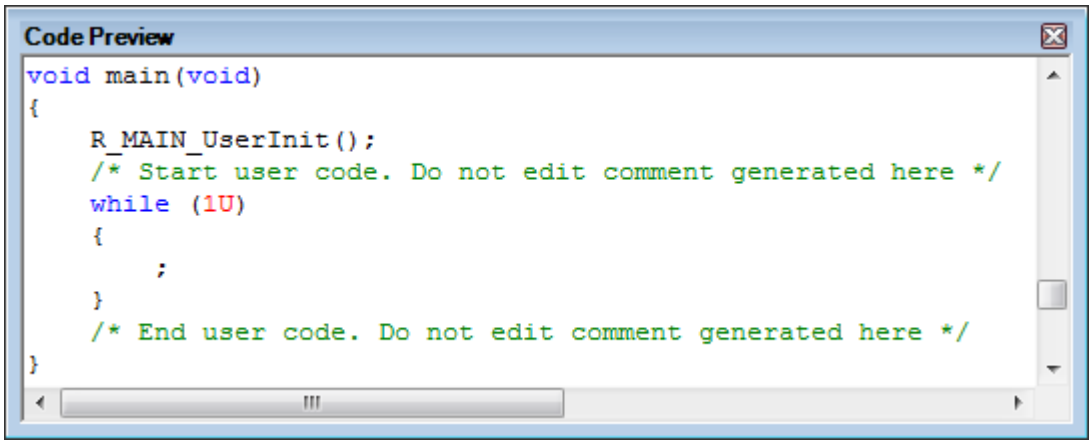


2.4 Confirm Source Code

Confirm the source code (device driver program) that reflects the information configured as described in "2.3 Enter Information".

To confirm the source code, use the Code Preview panel that opens by double-clicking [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Code Preview] >> Peripheral function node >> Source code node (>> API function node) in the Project Tree panel.

Figure 2-4. 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 Project Tree panel.

2. The following table displays the meaning of the color of the source code text displayed in the Code Preview panel.

Table 2-3. 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 Preview panel.

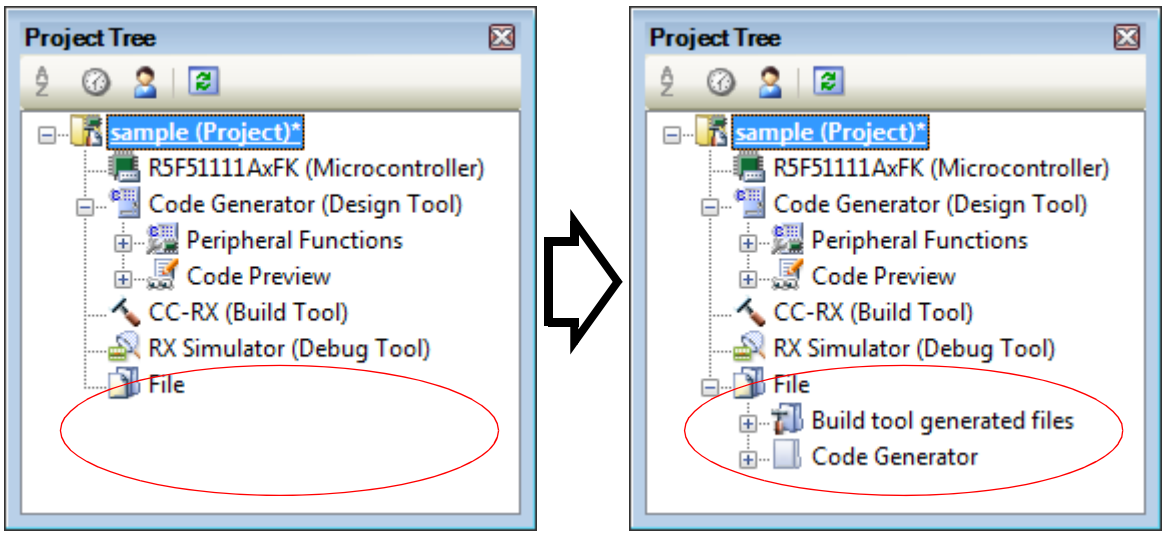
4. For some of the API functions, values such as the register value are calculated and finalized when the source code is generated (when the Generate Code button on the Peripheral Functions panel is pressed). For this reason, the source code displayed in this panel may not be the same as that would actually be generated.

2.5 Output Source Code

Output the source code (device driver program) by pressing the Generate Code button on the Peripheral Functions panel.

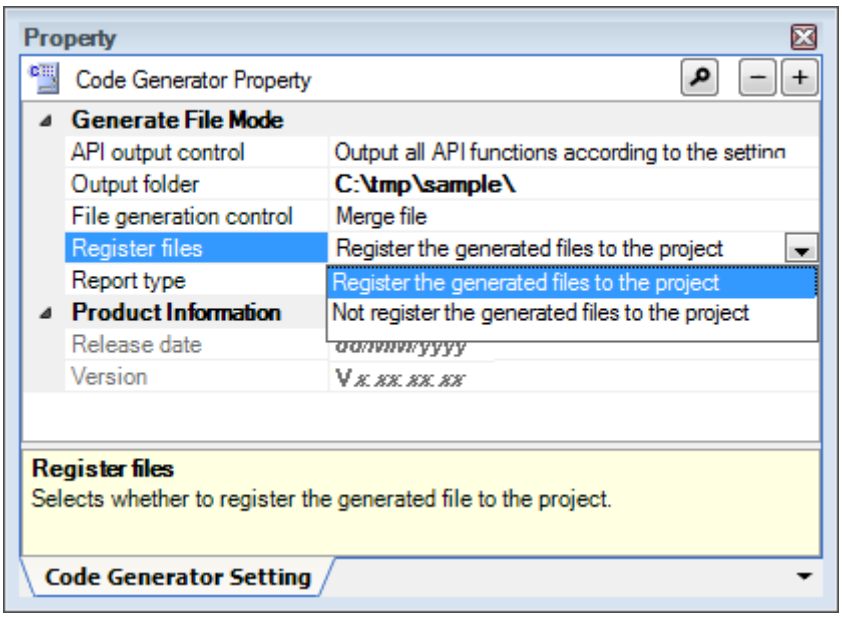
The destination folder for the source code is specified by clicking [Code Generator Setting] tab >> [Generate File Mode] >> [Output folder] in the Property panel.

Figure 2-5. 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 Generate Code button, you must open the Property panel, and under [Code Generator Setting] tab >> [Generate File Mode] >> [Register files], specify "Register the generated files to the project".

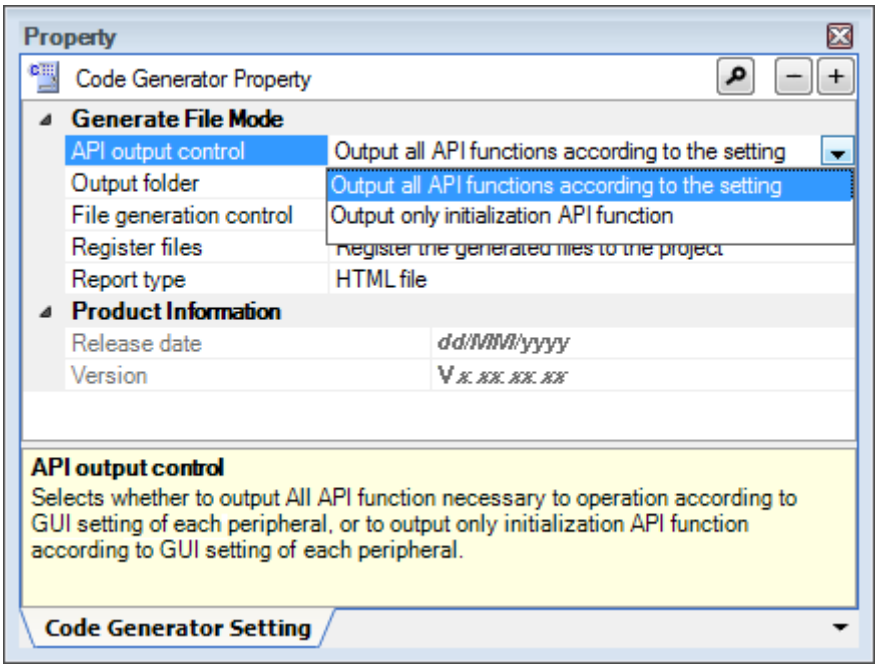
Figure 2-6. Configure Whether to Register



2.5.1 Set whether or not to generate source code

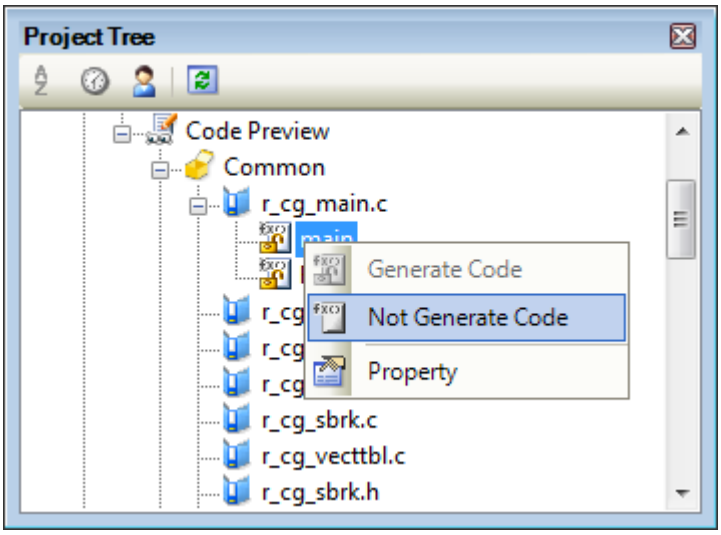
You can set the type of output API functions (all API functions or only initialization API functions) by selecting [Output all API functions according to the setting/Output only initialization API function] from [Code Generator Setting] tab >> [Generate File Mode] >> [API output control] in the Property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

Figure 2-7. Setting That Determines Type of API Functions



In the Code Generator, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Code Preview] >> Peripheral function node >> Source code node >> API function node in the Project Tree panel. "Setting That Determines Whether or Not to Generate Source Code" can be set in units of API functions by selecting "Generate Code/Not Generate Code" from the context menu, which is displayed by right clicking the mouse.

Figure 2-8. Setting That Determines Whether or Not to Generate Source Code



Remark "Setting That Determines Whether or Not to Generate Source Code" can be confirmed by the types of icons that are displayed immediately to the left of the API function nodes.

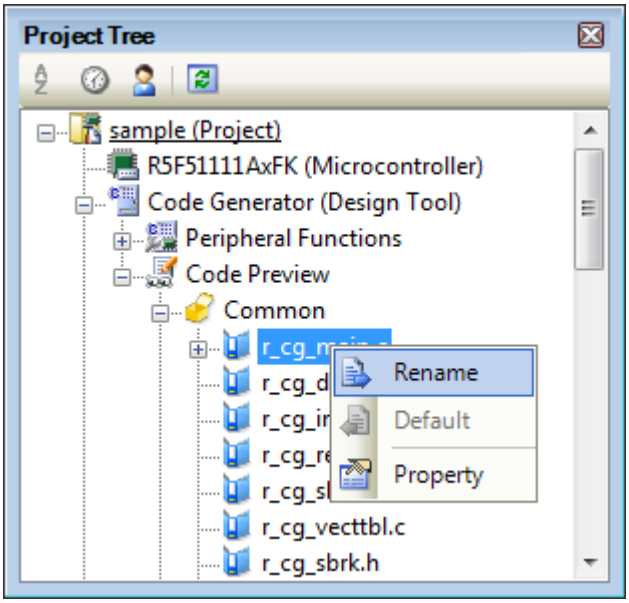
Table 2-4. Setting That Determines Whether or Not to Generate Source Code

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.

2.5.2 Change file name

In the Code Generator, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Code Preview] >> Peripheral function node >> Source code node in the Project Tree panel. The name of the file can be changed by selecting "Rename" from the context menu, which is displayed by right clicking the mouse.

Figure 2-9. Change File Name

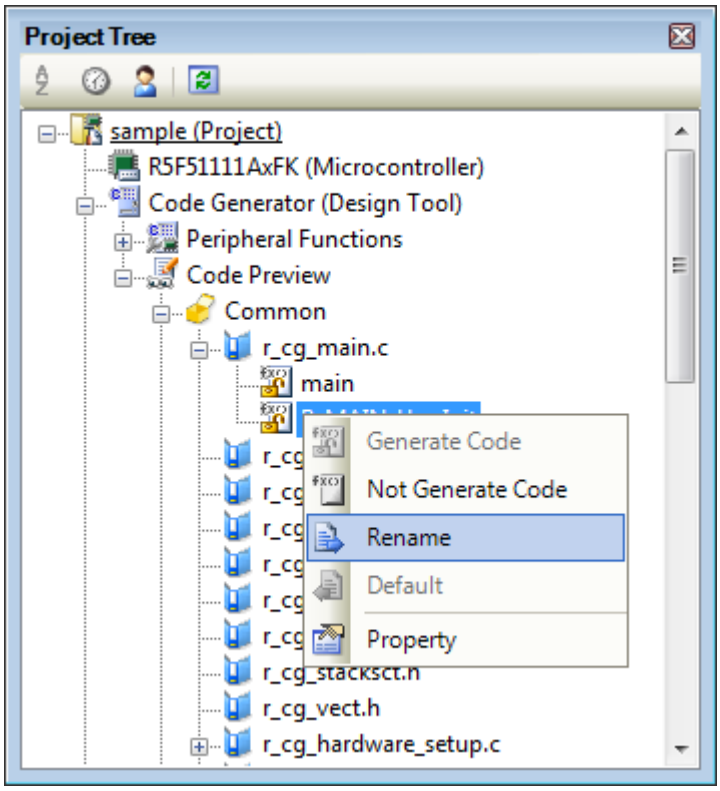


Remark To restore the default file name defined by the Code Generator, select [Default] from the context menu.

2.5.3 Change API function name

In the Code Generator, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Code Preview] >> Peripheral function node >> Source code node >> API function node in the Project Tree panel. The name of the API function can be changed by selecting "Rename" from the context menu, which is displayed by right clicking the mouse.

Figure 2-10. Change API Function Name



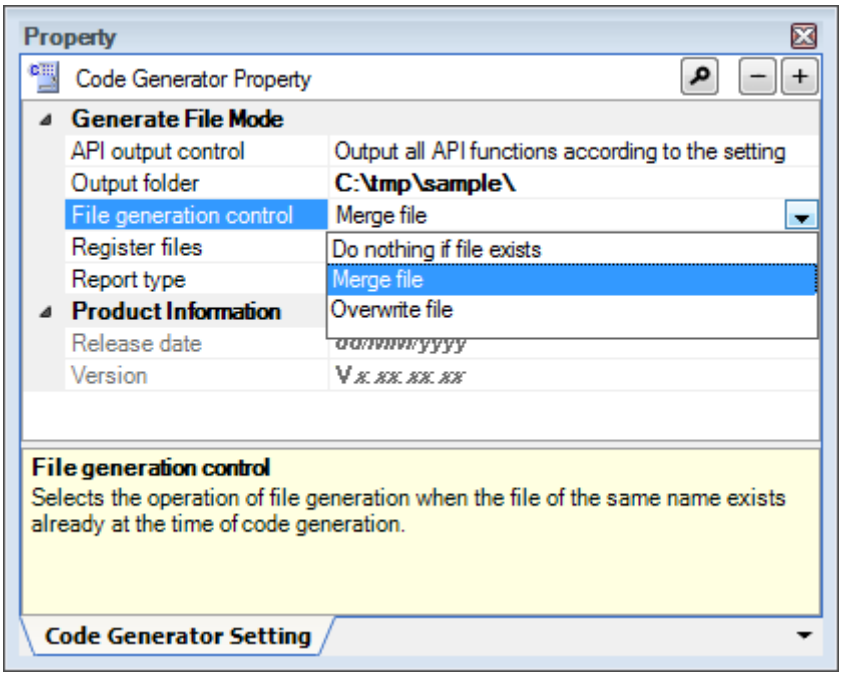
Remarks 1. To restore the default name of the API function defined by the Code Generator, select [Default] from the context menu.

2. Some API functions (main, etc.) can not be changed the API function name.

2.5.4 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 [Code Generator Setting] tab >> [Generate File Mode] >> [File generation control] in the Property panel.

Figure 2-11. Change Output Mode



The output mode is selected from the following three types.

Table 2-5. Output Mode of Source Code

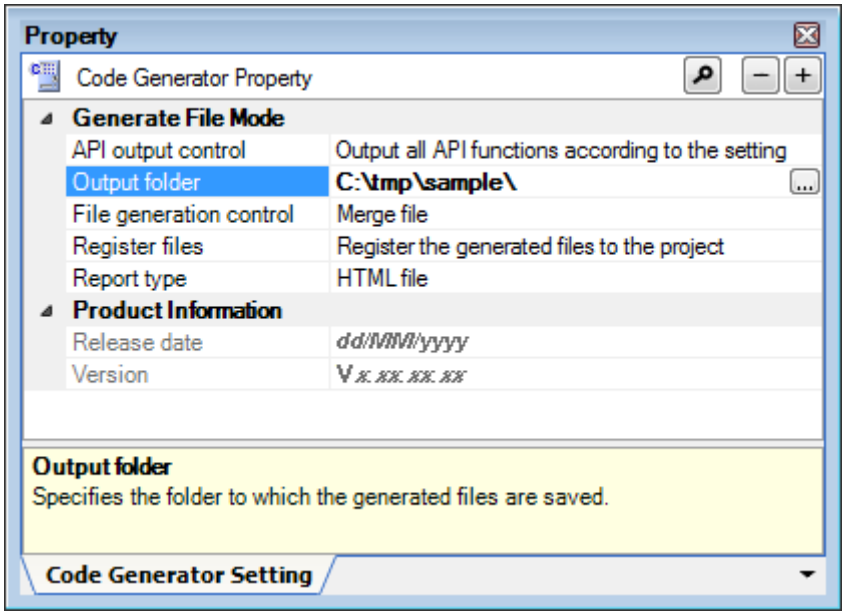
Output Mode	Outline
Do nothing if file exists	If a file with the same name exists, a new file will not be output.
Merge file	If a file with the same name exists, a new file is merged with the existing file. Only the section between "/* Start user code ... . Do not edit comment generated here */" and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

Remark Note that if the [Merge file] option is selected, the number of left braces ("{") and right braces ("}") must match in the parts to be merged. When the numbers do not match, processing for correct merging is not possible.

2.5.5 Change output destination folder

The Code Generator is used to change the output destination folder for the source code by selecting [Code Generator Setting] tab >> [Generate File Mode] >> [Output folder] in the Property panel.

Figure 2-12. Change Output Destination Folder



2.6 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 Peripheral Functions panel or Code Preview panel, then selecting [File] menu >> [Save Code Generator Report].

The destination folder for the report file is specified by clicking [Code Generator Setting] tab >> [Generate File Mode] >> [Output folder] in the Property panel.

Remarks 1. You can only use "Function" or "Macro" as a name of the report file.
See "2.6.1 Change output format" for details on the output format.

Table 2-6. Output Report Files

File Name	Outline
Function.*xxx*	A file that contains the information regarding the source code
Macro.*xxx*	A file that contains the information configured using Code Generator

2. The output mode of the report file is defined in "Overwrite file".

Figure 2-13. Output Example of Report File "Function" (HTML File)

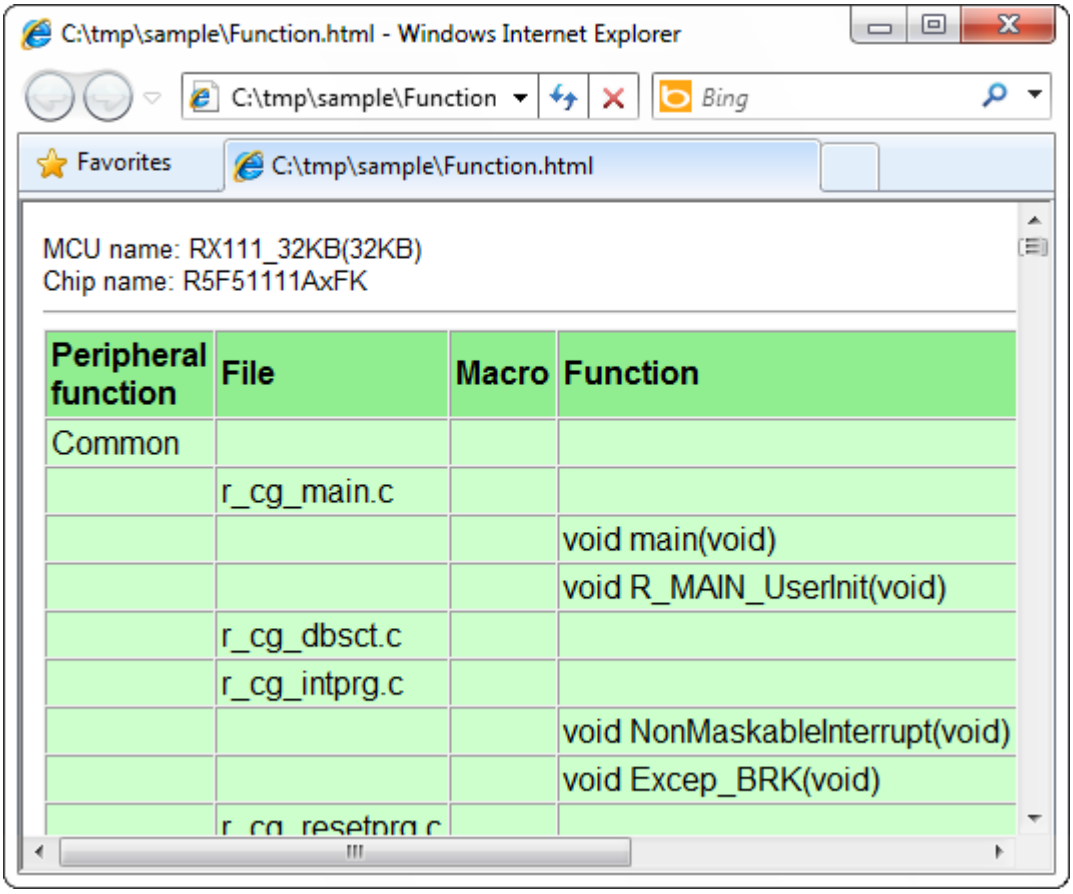
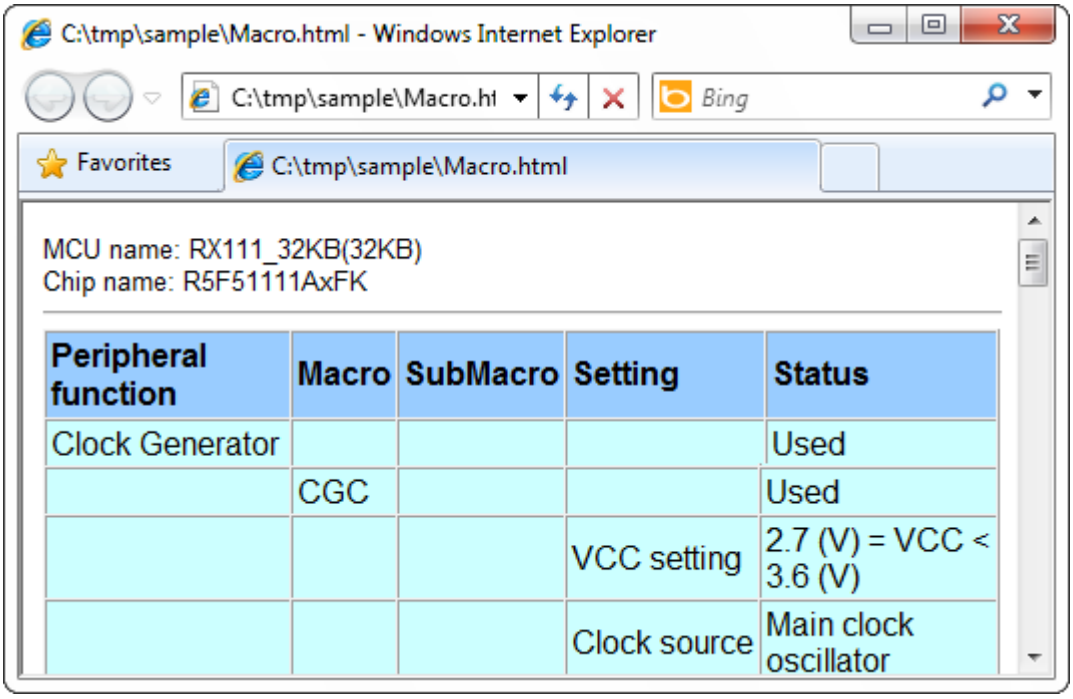


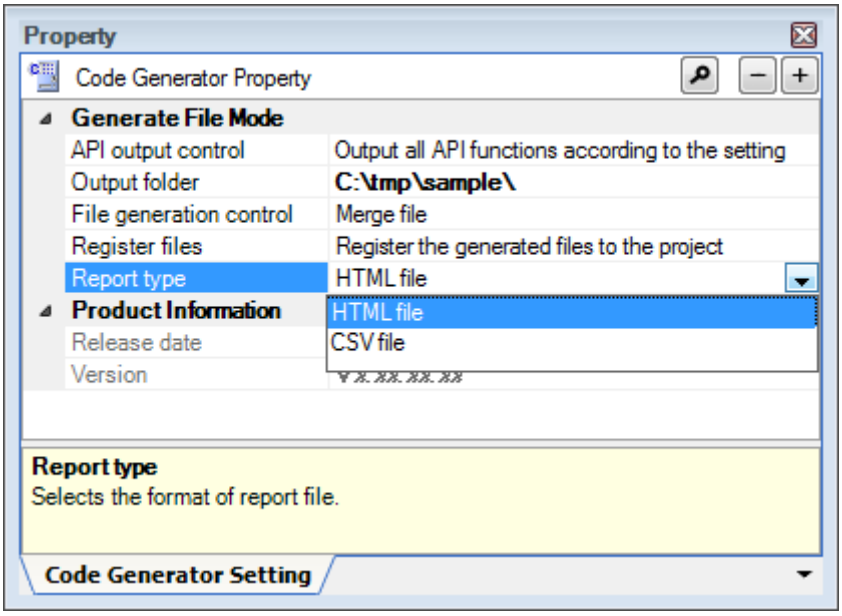
Figure 2-14. Output Example of Report File "Macro" (HTML File)



2.6.1 Change output format

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

Figure 2-15. Change Output Format



Remark The output format of the report file is selected from the two types shown below.

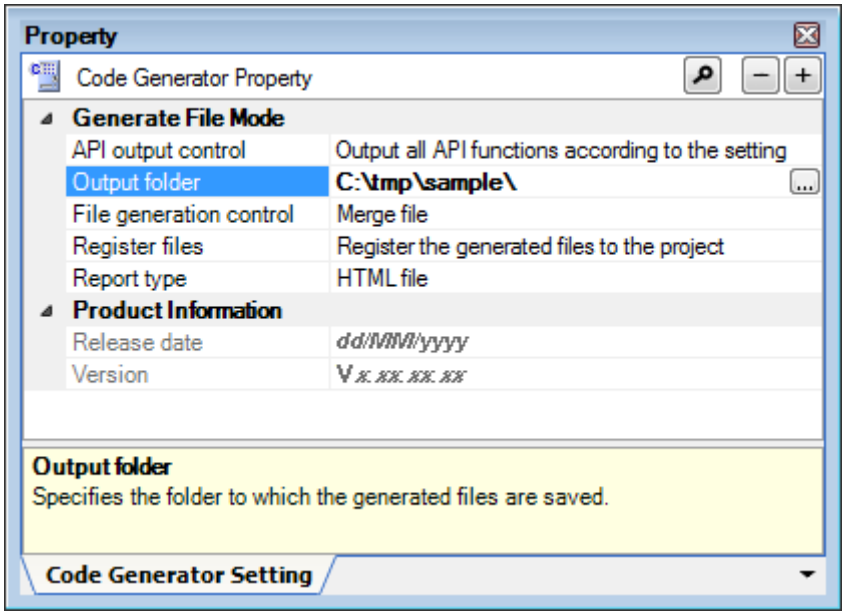
Table 2-7. Output Format of Report File

Report Type	Outline
HTML file	Outputs in the HTML format.
CSV file	Outputs in the CSV format.

2.6.2 Change output destination folder

The Code Generator is used to change the output destination folder for the report file by selecting [Code Generator Setting] tab >> [Generate File Mode] >> [Output folder] in the Property panel.

Figure 2-16. Change Output Destination Folder



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.

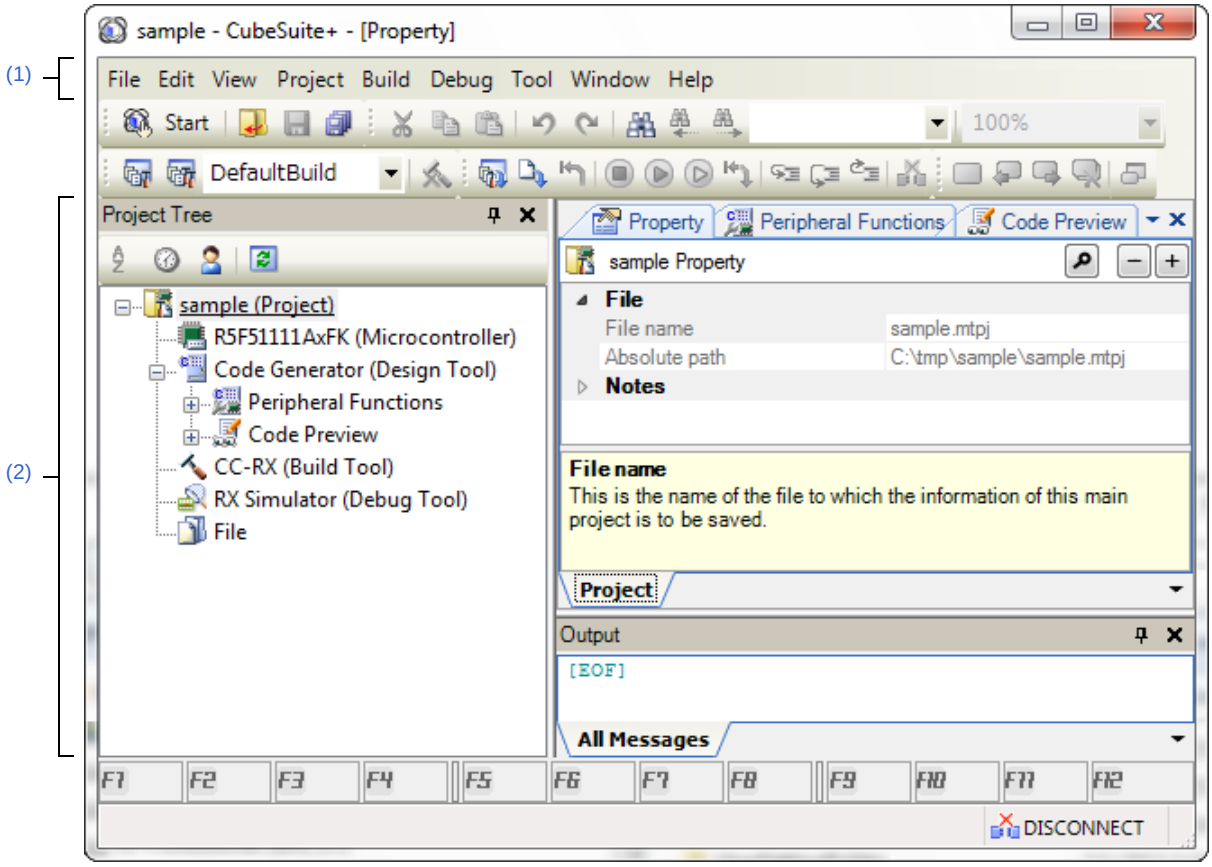
Table A-1. Window/Panel/Dialog Box List

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 for the node selected in the Project Tree panel.
Peripheral Functions panel	This panel allows you to configure the information necessary to control the peripheral functions (clock generation circuit, voltage detection circuit, etc.) provided.
Code Preview panel	This panel allows you to confirm the source code in accord with the settings of the Peripheral Functions panel.
Output panel	This panel displays operation logs for various components (design tool, build tool, etc.) provided by CubeSuite+.
Save As dialog box	This dialog box allows you to name and save a 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 Code Generator Report	Peripheral Functions panel/Code 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 file or CSV file) is selected by clicking [Code Generator Setting] tab >> [Generate File Mode] >> [Report type] in the Property panel. - The destination folder for the report file is specified by clicking [Code Generator Setting] tab >> [Generate File Mode] >> [Output folder] in the Property panel.
Save Output-*Tab Name*	Output panel-dedicated item Saves the message corresponding to the specified tab overwriting the existing file.
Save Output-*Tab Name* 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...	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) [View] menu

Project Tree	Project Tree panel-dedicated item Opens the Project Tree panel.
Property	Property panel-dedicated item Opens the Property panel.

Output	Output panel-dedicated item Opens the Output panel.	
Code Generator 2	The cascading menu shown below is displayed.	
	Peripheral Functions	Peripheral Functions panel-dedicated item Opens the Peripheral Functions panel.
	Code Preview	Code Preview panel-dedicated item Opens the Code Preview panel.

(d) [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 [Code Generator]panel	Peripheral Functions panel-dedicated item Displays the help of Peripheral Functions panel.
Open Help for [Code Generator Preview]panel	Code Preview panel-dedicated item Displays the help of Code 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.

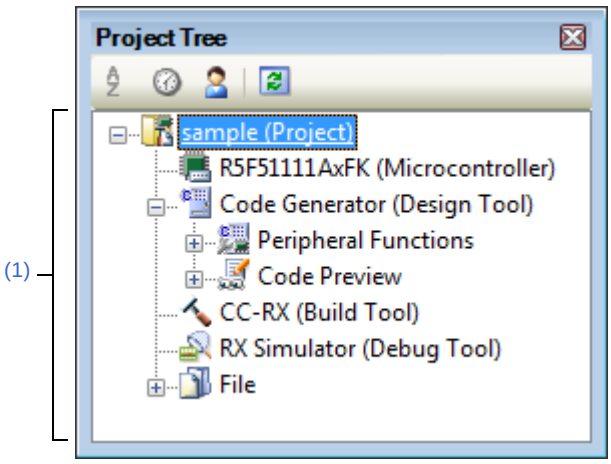
See the following sections for details on this area.

- Project Tree panel
- Property panel
- Peripheral Functions panel
- Code 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]
- [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) Code Generator (Design Tool)

The sub-nodes of this node are [Peripheral Functions] and [Code Preview].

<1> [Peripheral Functions]

The sub-node of this node is the peripheral function node for the peripheral functions (clock generation circuit, voltage detection circuit, etc.) supported by the target device.

Peripheral function node	Double-click on a peripheral function node or press the [Enter] key after selecting a peripheral function node to open the Peripheral Functions panel, which is used to make settings for control of the corresponding peripheral function.

Icons that are displayed immediately to the left of each peripheral function node have the meanings listed below.

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

<2> [Code Preview]

The sub-node of this node is the peripheral function node for the peripheral functions (clock generation circuit, voltage detection circuit, etc.) supported by the target device.

Peripheral function node	Double-click on a source code node/API function node in the level of the hierarchy below this node or select a source code node/API function node and press the [Enter] key to open the Code Preview panel, which is used to confirm that the source code corresponds to the settings in the Peripheral Functions panel.

Icons that are displayed immediately to the left of each peripheral function node have the meanings listed below.

	Operation in the corresponding Peripheral Functions panel has been carried out.
	Operation in the corresponding Peripheral Functions panel has not been carried out.

[Context menu]

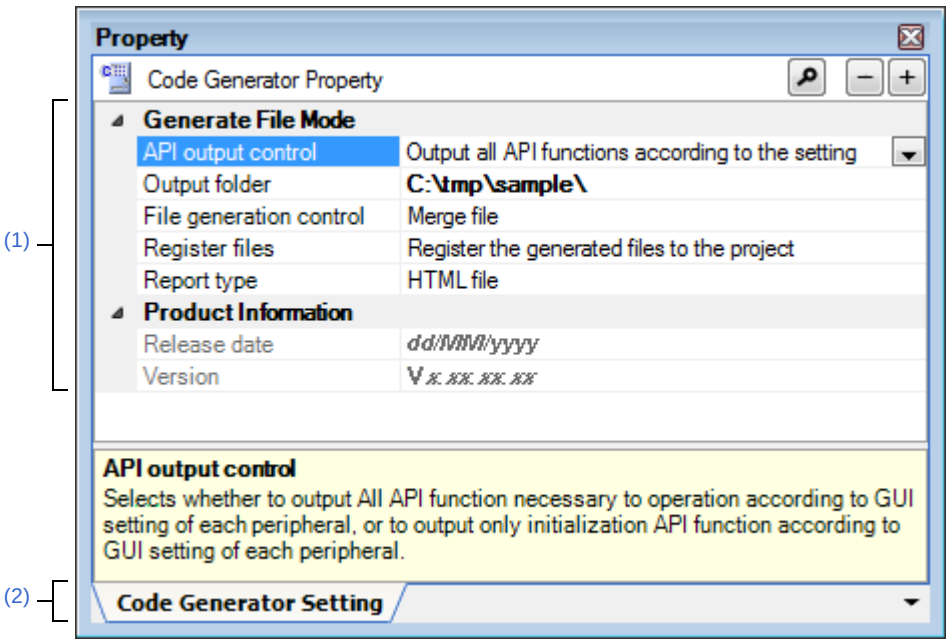
The following context menu items are displayed by right clicking the mouse.

Return to Reset Value	The default settings of the selected node are restored.
Property	Opens the Property panel corresponding to the selected node.

Property panel

This panel allows you to view the information on and change the setting for the node selected in the Project Tree panel.

Figure A-3. Property Panel



The following items are explained here.

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

[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.
- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Peripheral Functions] (>> Peripheral function node), and then select [Property] from the [View] menu.
- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Peripheral Functions] (>> Peripheral function node), and then select [Property] from the context menu.
- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Code Preview] (>> Peripheral function node >> Source code node >> API function node), and then select [Property] from the [View] menu.
- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Code Preview] (>> Peripheral function node >> Source code node >> API function node), and then select [Property] from the context menu.

Remarks 1. If this panel is already open, selecting a different [Code Generator (Design Tool)] in the Project Tree panel changes the content displayed accordingly.

2. If this panel is already open, selecting [Peripheral Functions] (>> Peripheral function node) in the Project Tree panel changes the content displayed to that corresponding to the selected node.

3. If this panel is already open, selecting [Code Preview] (>> Peripheral function node >> source code node >> API function node) in the Project Tree panel changes the content displayed to that corresponding to the selected node.

[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 selected in the Project Tree panel.

The content displayed in this area differs depending on the node selected in the Project Tree 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

In this panel, following tabs are contained (see the section explaining each tab for details on the display/setting on the tab).

- [Code Generator Setting] tab
- [Peripheral Function Information] tab (Product Information)
- [Peripheral Function Information] tab (Peripheral Function Information)
- [Code Preview Information] tab (Product Information)
- [Code Preview Information] tab (Peripheral Function Information)
- [Code Preview Setting] tab (File Information)
- [Code Preview Setting] tab (Function Information)

[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.

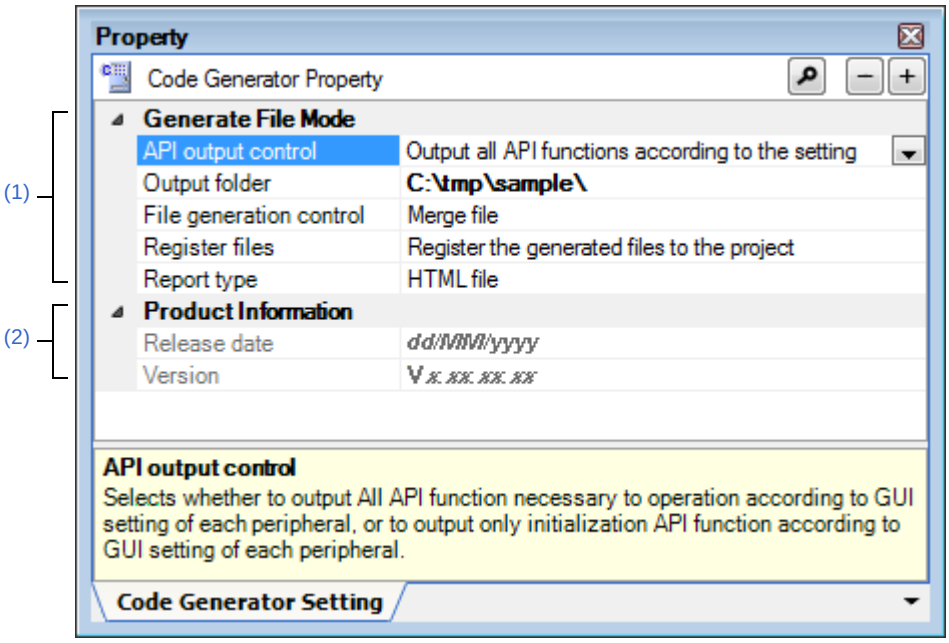
(2) While the item is not being edited

Property Reset to Default	The selected items are returned to their default settings.
Property Reset All to Default	All items displayed in this tab are returned to their default settings.

[Code Generator Setting] tab

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

Figure A-4. [Code Generator 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)], 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) [Generate File Mode] category

This area allows you to view the information (API output control, Output fo;der, File generation control, Register files and Report type) on and change the setting for the [Code Generator (Design Tool)] selected in the Project Tree panel.

API output control	Select the type of API functions to be output.	
	Output all API functions according to the setting	All API functions for the peripheral functions (clock generation circuit, voltage detection circuit, etc.) that is set for use in the Peripheral Functions panel are output.
	Output only initialization API function	Of the API functions for the peripheral functions (clock generation circuit, voltage detection circuit, etc.) that are set for use in the Peripheral Functions panel, only those relating to initialization are output.
Output folder	Inputs the output destination folder.	
File generation control	Click on this option to select the reaction to cases where a file having the same file name exists when the Generate Code button of the Peripheral Functions panel is clicked.	
	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.
Register files	Click on this option to select whether or not to register the output file in the project when the Generate Code button of the Peripheral Functions panel is clicked.	
	Register the generated files to the project	Registers the file.
	Not register the generated files to the project	Does not register the file.
Report type	Selects the output format for the report files (two files: Function and Macro) that are output when [Save Code Generator Report] is selected from the [File] menu.	
	HTML file	Outputs the files in the HTML format.
	CSV file	Outputs the files in the CSV format.

Remark Note that if the [Merge file] is selected in [File generate control], the number of left braces ("{") and right braces ("}") must match in the parts to be merged. When the numbers do not match, processing for correct merging is not possible.

(2) [Product Information] category

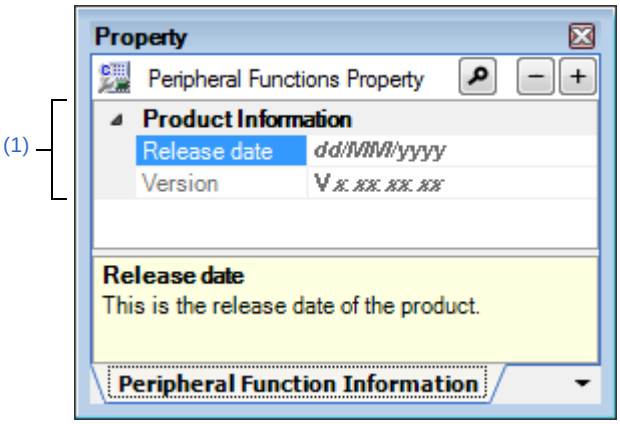
This area allows you to view the information (Release Date and Version) for the [Code Generator (Design Tool)] selected in the Project Tree panel.

Release Date	Displays the release date of the Code Generator (Design Tool).
Version	Displays the version number of the Code Generator (Design Tool).

[Peripheral Function Information] tab (Product Information)

This tab allows you to view the information (Product Information) for the [Peripheral Functions] selected in the Project Tree panel.

Figure A-5. [Peripheral Function Information] Tab (Product Information)



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

Remark If this panel is already open, selecting [Peripheral Functions] in the Project Tree panel changes the content displayed to that corresponding to the selected node.

[Description of each area]

(1) [Product Information] category

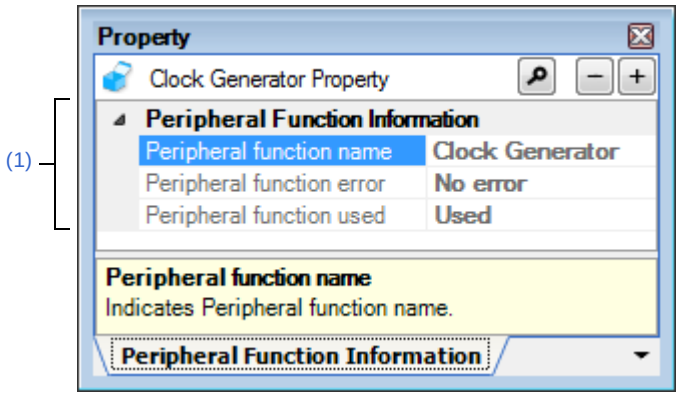
This area allows you to view the information (Release date and Version) for the [Peripheral Functions] selected in the Project Tree panel.

Release Date	Displays the release date of the Code Generator (Design Tool).
Version	Displays the version number of the Code Generator (Design Tool).

[Peripheral Function Information] tab (Peripheral Function Information)

This tab allows you to view the information (Peripheral Function Information) for the peripheral function node selected in the Project Tree panel.

Figure A-6. [Peripheral Function Information] Tab (Peripheral Function Information)



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

Remark If this panel is already open, selecting peripheral function node in the Project Tree panel changes the content displayed to that corresponding to the selected node.

[Description of each area]

(1) [Peripheral Function Information] category

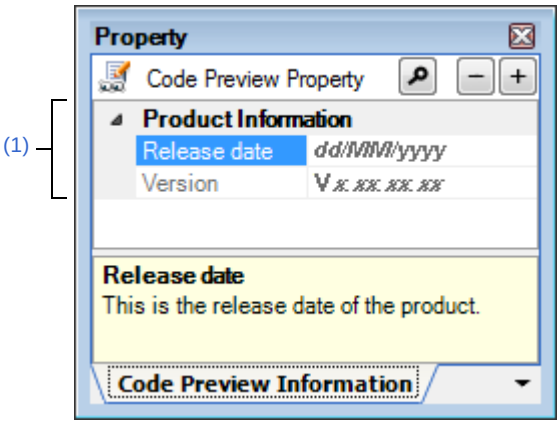
This area allows you to view the information (Peripheral function name, Peripheral function error and Peripheral function used) for the peripheral function node selected in the Project Tree panel.

Peripheral function name	Displays the name of the peripheral function.	
Peripheral function error	Displays whether or not the settings in the Peripheral Functions panel are correct.	
	No error	Illegal settings have not been detected.
	Input error	Illegal settings have been detected.
Peripheral function used	Indicates whether or not to use the peripheral function. Note that whether or not a function is to be used depends on the settings in the Peripheral Functions panel corresponding to the selected node.	
	Used	The peripheral function is to be used.
	No	The peripheral function is not to be used.

[Code Preview Information] tab (Product Information)

This tab allows you to view the information (Product Information) for the [Code Preview] selected in the Project Tree panel.

Figure A-7. [Code Preview Information] Tab (Product Information)



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

Remark If this panel is already open, selecting [Code Preview] in the Project Tree panel changes the content displayed to that corresponding to the selected node.

[Description of each area]

(1) [Product Information] category

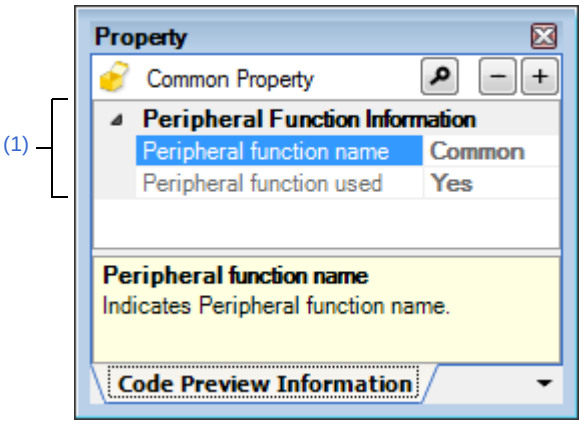
This area allows you to view the information (Release Date and Version) for the [Code Preview] selected in the Project Tree panel.

Release Date	Displays the release date of the Code Generator (Design Tool).
Version	Displays the version number of the Code Generator (Design Tool).

[Code Preview Information] tab (Peripheral Function Information)

This tab allows you to view the information (Peripheral Function Information) for the peripheral function node selected in the Project Tree panel.

Figure A-8. [Code Preview Information] Tab (Peripheral Function Information)



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

Remark If this panel is already open, selecting the peripheral function node in the Project Tree panel changes the content displayed to that corresponding to the selected node.

[Description of each area]

(1) [Peripheral Function Information] category

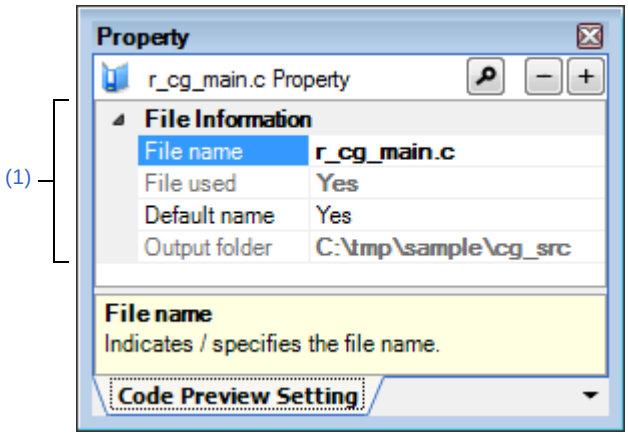
This area allows you to view the information (Peripheral function name and Peripheral function used) for the peripheral function node selected in the Project Tree panel.

Peripheral function name	Displays the name of the peripheral function.	
Peripheral function used	Indicates whether or not to use the peripheral function. Note that whether or not a function is to be used depends on the settings in the Peripheral Functions panel corresponding to the selected node.	
	Yes	The peripheral function is to be used.
	No	The peripheral function is not to be used.

[Code Preview Setting] tab (File Information)

This tab allows you to view the information (File Information) on and change the setting for the source code node selected in the Project Tree panel.

Figure A-9. [Code Preview Setting] Tab (File Information)



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)] >> [Code Preview] >> Peripheral function node >> Source code node, and then select [Property] from the [View] menu.
- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Code Preview] >> Peripheral function node >> Source code node, and then select [Property] from the context menu.

Remark If this panel is already open, selecting the source code node in the Project Tree panel changes the content displayed to that corresponding to the selected node.

[Description of each area]

(1) [File Information] category

This area allows you to view the information (Fine name, File used, Default name and Output folder) on and change the setting for the source code node selected in the Project Tree panel.

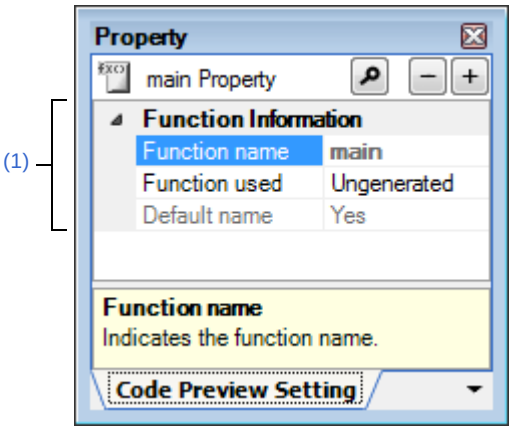
File name	Inputs the name of the file The name of the file can be changed by selecting [Rename] from the context menu after selecting the source code node in the Project Tree panel.	
File used	Indicates whether or not output to a file is to proceed when the Generate Code button in the Peripheral Functions panel is clicked. Note that whether or not this option is used depends on the settings in the Peripheral Functions panel corresponding to the selected node.	
	Yes	A file is output.
	No	A file is not output.

Default name	Selects whether or not to restore the default name of the file. Note that the default name of the file can be restored by selecting [Default] from the context menu after selecting the source code node in the Project Tree panel.	
	Yes	The default name is restored.
	No	The default name is not restored.
Output folder	Displays the output destination folder. Note that the output destination folder can be changed by using [Generate File Mode] >> [Output folder] in the [Code Generator Setting] tab.	

[Code Preview Setting] tab (Function Information)

This tab allows you to view the information (Function Information) on and change the setting for the API function node selected in the Project Tree panel.

Figure A-10. [Code Preview Setting] Tab (Function Information)



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)] >> [Code Preview] >> Peripheral function node >> Source code node >> API function node, and then select [Property] from the [View] menu.
- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Code Preview] >> Peripheral function node >> Source code node >> API function node, and then select [Property] from the context menu.

Remark If this panel is already open, selecting the API function node in the Project Tree panel changes the content displayed to that corresponding to the selected node.

[Description of each area]

(1) [Function Information] category

This area allows you to view the information (Function name, Function used and Default name) on and change the setting for the API function node selected in the Project Tree panel.

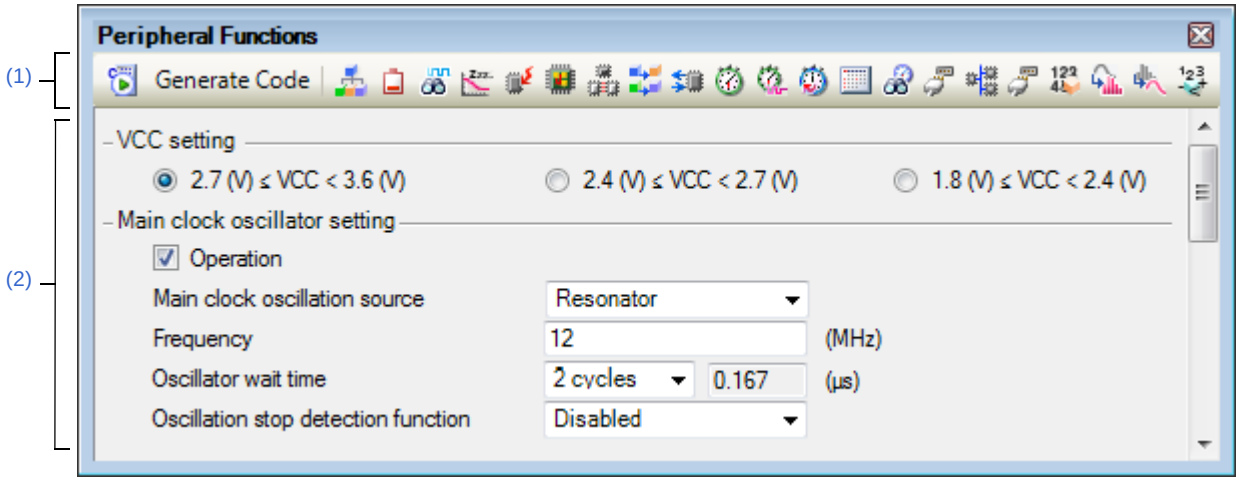
Function name	Inputs the name of the API function. Note that the name of the API function can be changed by selecting [Rename] from the context menu after selecting the API function node in the Project Tree panel.	
Function used	Selects whether or not to output the API function when the Generate Code button in the Peripheral Functions panel is clicked.	
	Generated	The API function is output.
	Ungenerated	The API function is not output.

Default name	Selects whether or not to restore the default name of the API function. Note that the default name of the API function can be restored by selecting [Default] from the context menu after selecting the source code node in the Project Tree panel.	
	Yes	The default name is restored.
	No	The default name is not restored.

Peripheral Functions panel

This panel allows you to configure the information necessary to control the peripheral functions (clock generation circuit, voltage detection circuit, etc.) provided.

Figure A-11. Peripheral Functions Panel



The following items are explained here.

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

[How to open]

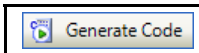
- On the Project Tree panel, double-click [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Peripheral Functions] (>> Peripheral function node).
- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Peripheral Functions] (>> Peripheral function node), and then press the [Enter] key.
- From the [View] menu >> [Code Generator 2], select [Peripheral Functions].

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 device is not supporting, peripheral functionbutton is not disokayed.

	Outputs the source code (device driver program) to the folder specified by selecting [Code Generator Setting] tab >> [Generate File Mode] >> [Output folder] in the Property panel.
 ,  , etc.	Changes the content displayed in the Information setting area to information required for controlling peripheral functions.

(2) Information setting area

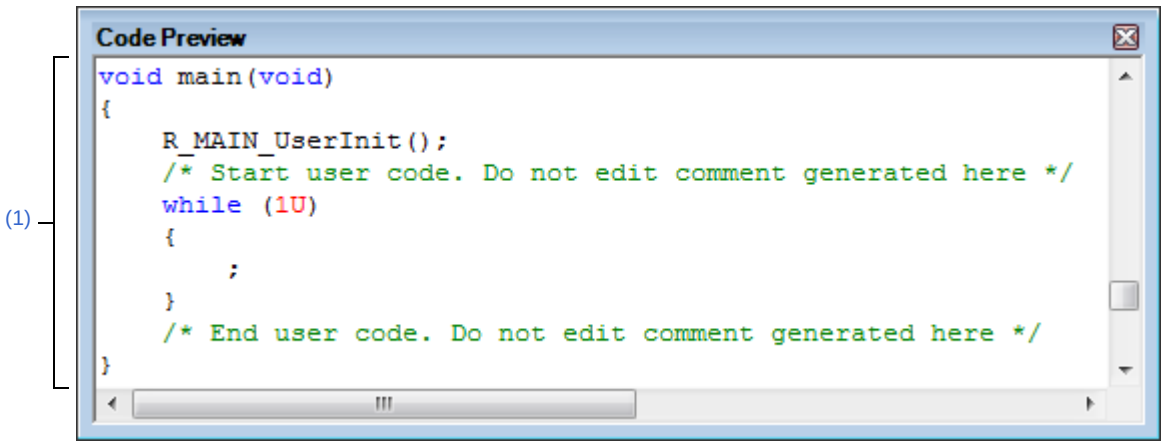
The content displayed in this area differs depending on the "peripheral function node" or "peripheral function button" selected or pressed when opening this panel.

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

Code Preview panel

This panel allows you to confirm the source code in accord with the settings of the Peripheral Functions panel.

Figure A-12. Code Preview Panel



The following items are explained here.

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

[How to open]

- On the Project Tree panel, double-click [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Code Preview] >> Peripheral function node >> Source code node (>> API function node).
- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> [Code Preview] >> Peripheral function node >> Source code node (>> API function mode), and then press the [Enter] key.
- From the [View] menu >> [Code Generator 2], select [Code Preview].

Remark If this panel is already open, double-clicking the source code node (>> API function node) changes the content displayed in the Source code display area to that corresponding to the selected node.

[Description of each area]

(1) Source code display area

This area allows you to confirm the source code (device driver program) that reflects the information configured in the Peripheral Functions panel.

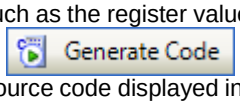
The following table displays the meaning of the color of the source code text displayed in this area.

Table A-2. Color of Source Code

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

Color	Outline
Gray	File name

Remarks 1. You cannot edit the source code within this panel.

2. For some of the API functions, values such as the register value are calculated and finalized when the source code is generated (when the 

button on the Peripheral Functions panel is pressed). For this reason, the source code displayed in this panel may not be the same as that would actually be generated.

[Context menu]

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

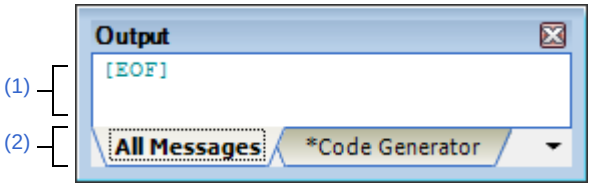
Copy	Sends the character string or lines selected with range selection to the clipboard.
Select All	Selects all the messages displayed on the Source code display area.

Output panel

This panel displays operation logs for various components (design tool, build 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 100% in the tool bar, or by operating the mouse wheel while holding down the [Ctrl] key.

Figure A-13. Output Panel



The following items are explained here.

- [How to open]
- [Description of each area]
- [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.
Note that the character colors/background colors of the message differ with the type of output message (and depend on the settings in the [General - Font and Color] category in the Option dialog box).

(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, 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, 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.

[Context menu]

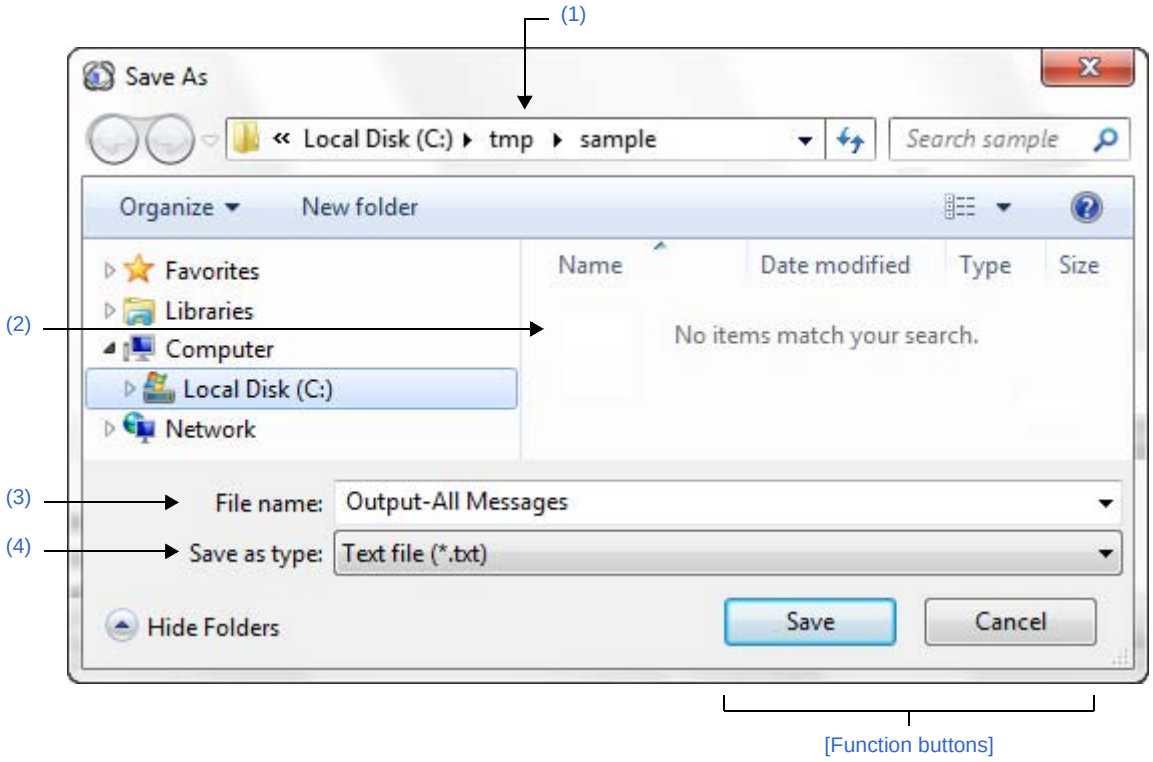
The following context menu items are displayed by right clicking the mouse.

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.
Tag Jump	Jumps to the caret line in the editor indicated by the message (file, line, and column).
Open Help for Message	Displays help for the message on the current caret location. This only applies to warning messages and error messages.

Save As dialog box

This dialog box allows you to name and save a file.

Figure A-14. 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 Output-*Tab Name*].
- From the [File] menu, select [Save Output-*Tab Name* As...].

[Description of each area]

(1) Folder location

This is for selection of the output destination folder (folder name).

(2) List of files

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

(3) [File name]

Specify the name of the file (file name).

(4) [Save as type]

Select the type of the file (file type).

[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 Folder location.
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 Description

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

Table B-1. Output File List

Peripheral Function	File Name	API Function Name
Common	r_cg_dbsct.c	-
	r_cg_hardware_setup.c	HardwareSetup R_Systeminit
	r_cg_intprg.c	NonMaskableInterrupt Excep_BRK
	r_cg_main.c	main R_MAIN_UserInit
	r_cg_resetprg.c	PowerON_Reset
	r_cg_sbrk.c	-
	r_cg_vecttbl.c	-
	r_cg_iodefine.h	-
	r_cg_macrodriver.h	-
	r_cg_sbrk.h	-
	r_cg_stacksct.h	-
	r_cg_userdefine.h	-
	r_cg_vect.h	-
Clock generation circuit	r_cg_cgc.c	R_CGC_Create R_CGC_Set_ClockMode
	r_cg_cgc_user.c	R_CGC_Create_UserInit r_cgc_oscillation_stop_interrupt
	r_cg_cgc.h	-
Voltage detection circuit (LVDA)	r_cg_lvd.c	R_LVDn_Create R_LVDn_Start R_LVDn_Stop
	r_cg_lvd_user.c	R_LVDn_Create_UserInit r_lvd_lvdn_interrupt
	r_cg_lvd.h	-

Peripheral Function	File Name	API Function Name
Clock frequency accuracy measurement circuit (CAC)	r_cg_cac.c	R_CAC_Create R_CAC_Start R_CAC_Stop
	r_cg_cac_user.c	R_CAC_Create_UserInit r_cac_mendf_interrupt r_cac_ferrf_interrupt r_cac_ovrf_interrupt
	r_cg_cac.h	-
Low power consumption	r_cg_lpc.c	R_LPC_Create R_LPC_AllModuleStop R_LPC_ChangeSleepModeRetrunClock R_LPC_Sleep R_LPC_DeepSleep R_LPC_SoftwareStandby R_LPC_ChangeOperationPowerControl
	r_cg_lpc_user.c	R_LPC_Create_UserInit
	r_cg_lpc.h	-
Interrupt controller (ICU)	r_cg_icu.c	R_ICU_Create R_ICU_IRQn_Start R_ICU_IRQn_Stop R_ICU_Software_Start R_ICU_Software_Stop R_ICU_SoftwareInterrupt_Generate
	r_cg_icu_user.c	R_ICU_Create_UserInit r_icu_irqn_interrupt r_icu_software_interrupt r_icu_nmi_interrupt
	r_cg_icu.h	-
Buses	r_cg_bsc.c	R_BSC_Create R_BSC_Error_Monitoring_Start R_BSC_Error_Monitoring_Stop
	r_cg_bsc_user.c	R_BSC_Create_UserInit r_bsc_buserr_interrupt
	r_cg_bsc.h	-
Data transfer controller (DTC)	r_cg_dtc.c	R_DTC_Create R_DTCm_Start R_DTCm_Stop
	r_cg_dtc_user.c	R_DTC_Create_UserInit
	r_cg_dtc.h	-

Peripheral Function	File Name	API Function Name
Event link controller (ELC)	r_cg_elc.c	R_ELC_Create R_ELC_Start R_ELC_Stop R_ELC_GenerateSoftwareEvent R_ELC_Set_PortBuffer1 R_ELC_Get_PortBuffer1
	r_cg_elc_user.c	R_ELC_Create_UserInit r_elc_elsr18i_interrupt
	r_cg_elc.h	-
I/O ports	r_cg_port.c	R_PORT_Create
	r_cg_port_user.c	R_PORT_Create_UserInit
	r_cg_port.h	-
Multi-function timer pulse unit 2 (MTU2)	r_cg_mtu2.c	R_MTU2_U0_Create R_MTU2_U0_Cn_Start R_MTU2_U0_Cn_Stop
	r_cg_mtu2_user.c	R_MTU2_U0_Create_UserInit r_mtu2_u0_tgimn_interrupt r_mtu2_u0_tcivn_interrupt r_mtu2_u0_tciun_interrupt
	r_cg_mtu2.h	-
Port output enable 2 (POE2)	r_cg_poe2.c	R_POE2_Create R_POE2_Start R_POE2_Stop
	r_cg_poe2_user.c	R_POE2_Create_UserInit r_poe2_oein_interrupt
	r_cg_poe2.h	-
Compare match timer (CMT)	r_cg_cmt.c	R_CMTn_Create R_CMTn_Start R_CMTn_Stop
	r_cg_cmt_user.c	R_CMTn_Create_UserInit r_cmt_cmin_interrupt
	r_cg_cmt.h	-

Peripheral Function	File Name	API Function Name
Realtime clock (RTCA)	r_cg_rtc.c	R_RTC_Create R_RTC_Set_CalendarAlarm R_RTC_Set_BinaryAlarm R_RTC_Set_ConstPeriodInterruptOn R_RTC_Set_ConstPeriodInterruptOff R_RTC_Set_CarryInterruptOn R_RTC_Set_CarryInterruptOff R_RTC_Set_RTCOUTOn R_RTC_Set_RTCOUTOff R_RTC_Start R_RTC_Stop R_RTC_Restart R_RTC_Set_CalendarCounterValue R_RTC_Get_CalendarCounterValue R_RTC_Set_BinaryCounterValue R_RTC_Get_BinaryCounterValue
	r_cg_rtc_user.c	R_RTC_Create_UserInit r_rtc_alm_interrupt r_rtc_prd_interrupt r_rtc_cup_interrupt
	r_cg_rtc.h	-
Independent watchdog timer (IWDT)	r_cg_iwdt.c	R_IWDT_Create R_IWDT_Restart
	r_cg_iwdt_user.c	R_IWDT_Create_UserInit r_iwdt_nmi_interrupt
	r_cg_iwdt.h	-

Peripheral Function	File Name	API Function Name
Serial communications interface (SCI)	r_cg_sci.c	R_SCIn_Create R_SCIn_Start R_SCIn_Stop R_SCIn_Serial_Send R_SCIn_Serial_Receive R_SCIn_Serial_Multiprocessor_Send R_SCIn_Serial_Multiprocessor_Receive R_SCIn_Serial_Send_Receive R_SCIn_SmartCard_Send R_SCIn_SmartCard_Receive R_SCIn_IIC_Master_Send R_SCIn_IIC_Master_Receive R_SCIn_SPI_Master_Send R_SCIn_SPI_Master_Send_Receive R_SCIn_SPI_Slave_Send R_SCIn_SPI_Slave_Send_Receive R_SCIn_IIC_StartCondition R_SCIn_IIC_StopCondition
	r_cg_sci_user.c	R_SCIn_Create_UserInit r_scin_transmitend_interrupt r_scin_transmit_interrupt r_scin_receiveend_interrupt r_scin_receiveerror_interrupt r_scin_callback_transmitend r_scin_callback_receiveend r_scin_callback_receiveerror
	r_cg_sci.h	-
I^2C bus interface (RIIC)	r_cg_riic.c	R_RIICn_Create R_RIICn_Start R_RIICn_Stop R_RIICn_Master_Send R_RIICn_Master_Receive R_RIICn_Slave_Send R_RIICn_Slave_Receive R_RIICn_StartCondition R_RIICn_StopCondition
	r_cg_riic_user.c	R_RIICn_Create_UserInit r_riicn_error_interrupt r_riicn_receive_interrupt r_riicn_transmit_interrupt r_riicn_transmitend_interrupt r_riicn_callback_receiveerror r_riicn_callback_transmitend r_riicn_callback_receiveend
	r_cg_riic.h	-

Peripheral Function	File Name	API Function Name
Serial peripheral interface (RSPI)	r_cg_rspi.c	R_RSPIn_Create R_RSPIn_Start R_RSPIn_Stop R_RSPIn_LoopBackReversed R_RSPIn_LoopBackDirect R_RSPIn_LoopBackDisable R_RSPIn_Send R_RSPIn_Send_Receive
	r_cg_rspi_user.c	R_RSPIn_Create_UserInit r_rspin_receive_interrupt r_rspin_transmit_interrupt r_rspin_error_interrupt r_rspin_idle_interrupt r_rspin_callback_receiveend r_rspin_callback_error r_rspin_callback_transmitend
	r_cg_rspi.h	-
CRC calculator (CRC)	r_cg_crc.c	R_CRC_SetCRC8 R_CRC_SetCRC16 R_CRC_SetCCITT R_CRC_Input_Data R_CRC_Get_Result
	r_cg_crc.h	-
12-bit A/D converter (S12AD)	r_cg_s12adc.c	R_S12ADb_Create R_S12ADb_Start R_S12ADb_Stop R_S12ADb_Get_ValueResult
	r_cg_s12adc_user.c	R_S12ADb_Create_UserInit r_s12ad_interrupt r_s12ad_groupb_interrupt
	r_cg_s12adc.h	-
D/A converter (DA)	r_cg_dac.c	R_DAC_Create R_DACm_Start R_DACm_Stop R_DACm_Set_ConversionValue
	r_cg_dac_user.c	R_DAC_Create_UserInit
	r_cg_dac.h	-

Peripheral Function	File Name	API Function Name
Data operation circuit (DOC)	r_cg_doc.c	R_DOC_Create R_DOC_SetMode R_DOC_WriteData R_DOC_GetResult R_DOC_ClearFlag
	r_cg_doc_user.c	R_DOC_Create_UserInit r_doc_dopcf_interrupt
	r_cg_doc.h	-

APPENDIX C API FUNCTIONS

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

C.1 Overview

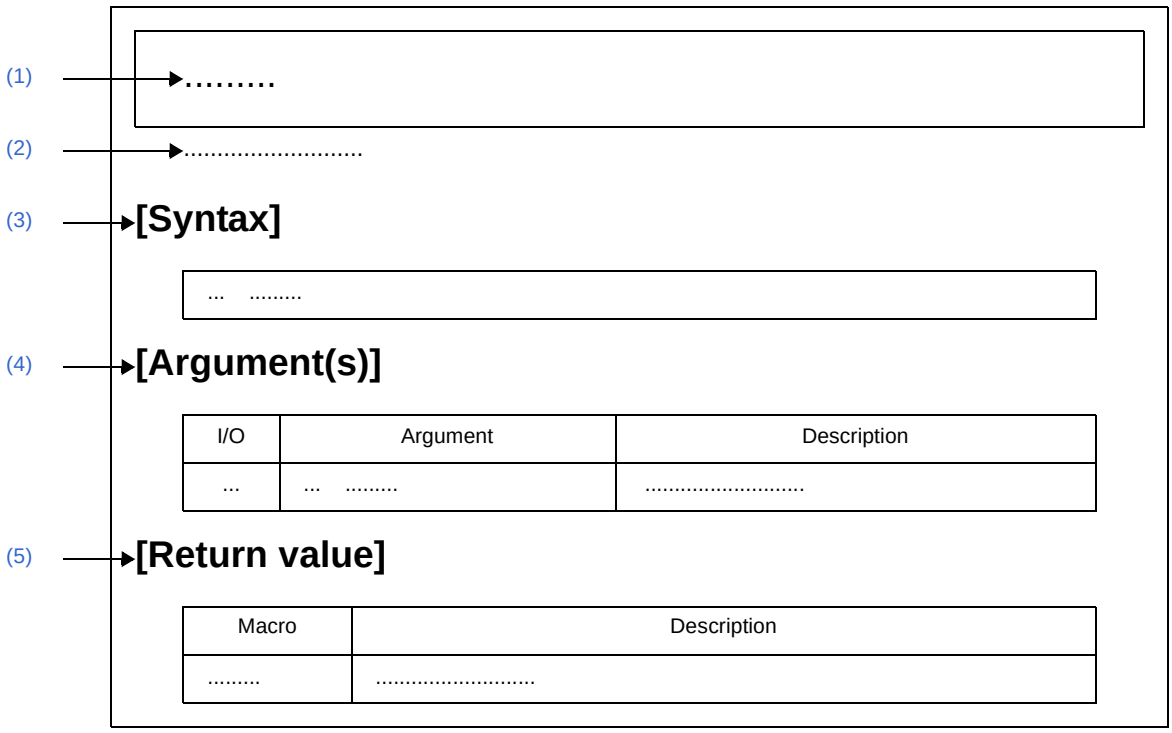
Below are the naming conventions for API functions output by the Code Generator.

- 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 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) [Syntax]

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

(4) [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

(5) [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

C.2.1 Common

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

Table C-1. API Functions: [Common]

API Function Name	Function
PowerON_Reset	Performs processing in response to the reset.
NonMaskableInterrupt	Performs processing in response to the non-maskable interrupt.
Excep_BRK	Performs processing in response to the unconditional trap.
HardwareSetup	Performs initialization necessary to control the various hardwares.
R_Systeminit	Performs initialization necessary to control the various peripheral functions.
main	This is a main function.
R_MAIN_UserInit	Performs user-defined initialization.

PowerON_Reset

Performs processing in response to the reset.

Remark This API function is called to run interrupt processing for an internal reset by the power-on reset circuit.

[Syntax]

```
void    PowerON_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

NonMaskableInterrupt

Performs processing in response to the non-maskable interrupt.

Remark This API function is called to run interrupt processing for the non-maskable interrupt.

[Syntax]

```
void    NonMaskableInterrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

Excep_BRK

Performs processing in response to the unconditional trap.

Remark This API function is called to run interrupt processing for an unconditional trap.

[Syntax]

```
void    Excep_BRK ( void );
```

[Argument(s)]

None.

[Return value]

None.

HardwareSetup

Performs initialization necessary to control the various hardwares.

Remark This API function is called as the PowerON_Reset callback routine.

[Syntax]

```
void    HardwareSetup ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_Systeminit

Performs initialization necessary to control the various peripheral functions.

Remark This API function is called as the HardwareSetup callback routine.

[Syntax]

```
void    R_Systeminit ( void );
```

[Argument(s)]

None.

[Return value]

None.

main

This is a main function.

Remark This API function is called as the PowerON_Reset callback routine.

[Syntax]

```
void    main ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MAIN_UserInit

Performs user-defined initialization.

Remark This API function is called as the main callback routine.

[Syntax]

```
void    R_MAIN_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.2.2 Clock generation circuit

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

Table C-2. API Functions: [Clock Generation Circuit]

API Function Name	Function
R_CGC_Create	Performs initialization required to control the clock generation circuit.
R_CGC_Create_UserInit	Performs user-defined initialization relating to the clock generation circuit.
r_cgc_oscillation_stop_interrupt	Performs processing in response to the osillation stop detection interrupt.
R_CGC_Set_ClockMode	Sets the clock source.

R_CGC_Create

Performs initialization required to control the clock generation circuit.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_CGC_Create_UserInit

Performs user-defined initialization relating to the clock generation circuit.

Remark This API function is called as the R_CGC_Create callback routine.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_cgc_oscillation_stop_interrupt

Performs processing in response to the oscillation stop detection interrupt.

Remark This API function is called to run interrupt processing for the oscillation stop detection interrupt, which is generated when the clock generation circuit detects oscillation by the main clock having stopped.

[Syntax]

```
static  void    r_cgc_oscillation_stop_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_Set_ClockMode

Sets the clock source.

[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 source type MAINCLK: Main clock oscillator SUBCLK: Sub-clock oscillator PLLCLK: PLL circuit HOCO: High-speed on-chip oscillator LOCO: Low-speed on-chip oscillator

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)
MD_ARGERROR	Invalid argument *mode* specification

C.2.3 Voltage detection circuit (LVDA)

Below is a list of API functions output by the Code Generator for voltage detection circuit use.

Table C-3. API Functions: [Voltage Detection Circuit]

API Function Name	Function
R_LVDn_Create	Performs initialization necessary to control the voltage detection circuit.
R_LVDn_Create_UserInit	Performs user-defined initialization relating to the voltage detection circuit.
r_lvd_lvdn_interrupt	Performs processing in response to the voltage monitoring *n* interrupt.
R_LVDn_Start	Starts voltage monitoring (when in interrupt mode, and interrupt & reset mode).
R_LVDn_Stop	Ends voltage monitoring (when in interrupt mode, and interrupt & reset mode).

R_LVD*n*_Create

Performs initialization necessary to control the voltage detection circuit.

[Syntax]

```
void    R_LVDn_Create ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

R_LVD*n*_Create_UserInit

Performs user-defined initialization relating to the voltage detection circuit.

Remark This API function is called as the R_LVDn_Create callback routine.

[Syntax]

```
void    R_LVDn_Create_UserInit ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

r_lvd_lvd*n*_interrupt

Performs processing in response to the voltage monitoring *n* interrupt.

Remark This API function is called to run interrupt processing for the voltage monitoring *n* interrupt, which is generated when the voltage detection circuit detects the voltage being dropped.

[Syntax]

```
static  void    r_lvd_lvdn_interrupt ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

R_LVD*n*_Start

Starts voltage monitoring (when in interrupt mode, and interrupt & reset mode).

[Syntax]

```
void    R_LVDn_Start ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

R_LVD*n*_Stop

Ends voltage monitoring (when in interrupt mode, and interrupt & reset mode).

[Syntax]

```
void    R_LVDn_Stop ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

C.2.4 Clock frequency accuracy measurement circuit (CAC)

Below is a list of API functions output by the Code Generator for clock frequency accuracy measurement circuit use.

Table C-4. API Functions: [Clock Frequency Accuracy Measurement Circuit]

API Function Name	Function
R_CAC_Create	Performs initialization necessary to control the clock frequency accuracy measurement circuit.
R_CAC_Create_UserInit	Performs user-defined initialization relating to the clock frequency accuracy measurement circuit.
r_cac_mendf_interrupt	Performs processing in response to the measurement end interrupt.
r_cac_ferrf_interrupt	Performs processing in response to the frequency error interrupt.
r_cac_ovrf_interrupt	Performs processing in response to the overflow interrupt.
R_CAC_Start	Starts measurement of the accuracy of the clock frequency.
R_CAC_Stop	Ends measurement of the accuracy of the clock frequency.

R_CAC_Create

Performs initialization necessary to control the clock frequency accuracy measurement circuit.

[Syntax]

```
void    R_CAC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CAC_Create_UserInit

Performs user-defined initialization relating to the clock frequency accuracy measurement circuit.

Remark This API function is called as the R_CAC_Create callback routine.

[Syntax]

```
void    R_CAC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cac_mendf_interrupt

Performs processing in response to the measurement end interrupt.

Remark This API function is called to run interrupt processing for the measurement end interrupt, which is generated when the clock frequency accuracy measurement circuit detects the valid edge of the reference signal.

[Syntax]

```
static  void    r_cac_mendf_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cac_ferrf_interrupt

Performs processing in response to the frequency error interrupt.

Remark This API function is called to run interrupt processing for the frequency error interrupt, which is generated when the clock frequency is not in the allowed range (from the minimum to the maximum value).

[Syntax]

```
static  void    r_cac_ferrf_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cac_ovrf_interrupt

Performs processing in response to the overflow interrupt.

Remark This API function is called to run interrupt processing for the overflow interrupt, which is generated when the counter overflows.

[Syntax]

```
static  void    r_cac_ovrf_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CAC_Start

Starts measurement of the accuracy of the clock frequency.

[Syntax]

```
void    R_CAC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CAC_Stop

Ends measurement of the accuracy of the clock frequency.

[Syntax]

```
void    R_CAC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.2.5 Low power consumption

Below is a list of API functions output by the Code Generator for low power consumption use.

Table C-5. API Functions: [Low Power Consumption]

API Function Name	Function
R_LPC_Create	Performs initialization required to control the low power consumption.
R_LPC_Create_UserInit	Performs user-defined initialization relating to the low power consumption.
R_LPC_AllModuleStop	Stops the clock for all modules.
R_LPC_ChangeSleepModeRetrunClock	Sets the clock source that is selected following release from sleep mode.
R_LPC_Sleep	Transits the low power consumption mode of the MCU to the sleep mode.
R_LPC_DeepSleep	Transits the low power consumption mode of the MCU to the deep sleep mode.
R_LPC_SoftwareStandby	Transits the low power consumption mode of the MCU to the software standby mode.
R_LPC_ChangeOperationPowerControl	Changes the operating power control mode of the MCU.

R_LPC_Create

Performs initialization required to control the low power consumption.

[Syntax]

```
void    R_LPC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LPC_Create_UserInit

Performs user-defined initialization relating to the low power consumption.

Remark This API function is called as the R_LPC_Create callback routine.

[Syntax]

```
void    R_LPC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LPC_AllModuleStop

Stops the clock for all modules.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_LPC_AllModuleStop ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_ChangeSleepModeRetrunClock

Sets the clock source that is selected following release from sleep mode.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_lpc.h"
MD_STATUS   R_LPC_ChangeSleepModeReturnClock ( return_clock_t clock );
```

[Argument(s)]

I/O	Argument	Description
I	`return_clock_t clock;`	Clock source type RETURN_LOCO: Low-speed on-chip oscillator RETURN_HOCO: High-speed on-chip oscillator RETURN_MAIN_CLOCK: Main clock oscillator RETURN_DISABLE: Switching of the clock source does not proceed.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Change to the low-speed operating mode ended abnormally.
MD_ARGERROR	Invalid argument *clock* specification

R_LPC_Sleep

Transits the low power consumption mode of the MCU to the sleep mode.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_LPC_Sleep ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_DeepSleep

Transits the low power consumption mode of the MCU to the deep sleep mode.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_LPC_DeepSleep ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_SoftwareStandby

Transits the low power consumption mode of the MCU to the software standby mode.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_LPC_SoftwareStandby ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_ChangeOperationPowerControl

Changes the operating power control mode of the MCU.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_lpc.h"
MD_STATUS   R_LPC_ChangeOperationPowerControl ( operaring_mode_t mode );
```

[Argument(s)]

I/O	Argument	Description
I	`operating_mode_t` *mode;*	Operating power control mode type HIGH_SPEED: High-speed operating mode MIDDLE_SPEED: Middle-speed operating mode LOW_SPEED: Low-speed operating mode

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Change to the low-speed operating mode ended abnormally.
MD_ERROR2	Change to the middle-speed operating mode is ended abnormally.
MD_ARGERROR	Invalid argument *mode* specification

C.2.6 Interrupt controller (ICU)

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

Table C-6. API Functions: [Interrupt Controller]

API Function Name	Function
R_ICU_Create	Performs initialization necessary to control the interrupt controller.
R_ICU_Create_UserInit	Performs user-defined initialization relating to the interrupt controller.
r_icu_irqn_interrupt	Performs processing in response to the external pin interrupts.
r_icu_software_interrupt	Performs processing in response to the software interrupt.
r_icu_nmi_interrupt	Performs processing in response to the NMI pin interrupt.
R_ICU_IRQn_Start	Allows detection of the external pin interrupt.
R_ICU_IRQn_Stop	Prohibits detection of the external pin interrupt.
R_ICU_Software_Start	Allows detection of the software interrupt.
R_ICU_Software_Stop	Prohibits detection of the software interrupt.
R_ICU_SoftwareInterrupt_Generate	Generates the software interrupt.

R_ICU_Create

Performs initialization necessary to control the interrupt controller.

[Syntax]

```
void    R_ICU_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_Create_UserInit

Performs user-defined initialization relating to the interrupt controller.

Remark This API function is called as the R_ICU_Create callback routine.

[Syntax]

```
void    R_ICU_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_icu_irq*n*_interrupt

Performs processing in response to the external pin interrupts.

Remark This API function is called to run interrupt processing for the external pin interrupts.

[Syntax]

```
static  void    r_icu_irqn_interrupt ( void );
```

Remark *n* is the source number.

[Argument(s)]

None.

[Return value]

None.

r_icu_software_interrupt

Performs processing in response to the software interrupt.

Remark This API function is called to run the interrupt processing for the software interrupt, which is generated in response to calling of R_ICU_SoftwareInterrupt_Generate.

[Syntax]

```
static  void    r_icu_software_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_icu_nmi_interrupt

Performs processing in response to the NMI pin interrupt.

Remark This API function is called to run interrupt processing for the NMI pin interrupt.

[Syntax]

```
static  void    r_icu_nmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_IRQ*n*_Start

Allows detection of the external pin interrupt.

[Syntax]

```
void    R_ICU_IRQn_Start ( void );
```

Remark *n* is the source number.

[Argument(s)]

None.

[Return value]

None.

R_ICU_IRQ*n*_Stop

Prohibits detection of the external pin interrupt.

[Syntax]

```
void    R_ICU_IRQn_Stop ( void );
```

Remark *n* is the source number.

[Argument(s)]

None.

[Return value]

None.

R_ICU_Software_Start

Allows detection of the software interrupt.

[Syntax]

```
void    R_ICU_Software_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_Software_Stop

Prohibits detection of the software interrupt.

[Syntax]

```
void    R_ICU_Software_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_SoftwareInterrupt_Generate

Generates the software interrupt.

Remark r_icu_software_interrupt is called in response to calling od this API function.

[Syntax]

```
void    R_ICU_SoftwareInterrupt_Generate ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.2.7 Buses

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

Table C-7. API Functions: [Buses]

API Function Name	Function
R_BSC_Create	Performs initialization necessary to control the buses.
R_BSC_Create_UserInit	Performs user-defined initialization relating to the buses.
r_bsc_buserr_interrupt	Performs processing in response to the bus error (illegal address access).
R_BSC_Error_Monitoring_Start	Allows the detection of bus errors (illegal address access).
R_BSC_Error_Monitoring_Stop	Prohibits the detection of bus errors (illegal address access).

R_BSC_Create

Performs initialization necessary to control the buses.

[Syntax]

```
void    R_BSC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_Create_UserInit

Performs user-defined initialization relating to the buses.

Remark This API function is called as the R_BSC_Create callback routine.

[Syntax]

```
void    R_BSC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_bsc_buserr_interrupt

Performs processing in response to the bus error (illegal address access).

Remarks 1. This API function is called to run interrupt processing for a bus error (illegal address access), which is generated through access by the processing program to a location within an illegal address range.

2. The bus master that caused the bus error can be confirmed by reading the MST bit of bus error status register 1 (BERSR1) from within this API function.

3. The illegal address (high-order 13 bits) that caused the bus error can be confirmed by reading the ADDR bit of bus error status register 2 (BERSR2) from within this API function.

[Syntax]

```
static  void    r_bsc_buserr_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_Error_Monitoring_Start

Allows the detection of bus errors (illegal address access).

[Syntax]

```
void    R_BSC_Error_Monitoring_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_Error_Monitoring_Stop

Prohibits the detection of bus errors (illegal address access).

[Syntax]

```
void    R_BSC_Error_Monitoring_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.2.8 Data transfer controller (DTC)

Below is a list of API functions output by the Code Generator for data transfer controller use.

Table C-8. API Functions: [Data Transfer Controller]

API Function Name	Function
R_DTC_Create	Performs initialization necessary to control the data transfer controller.
R_DTC_Create_UserInit	Performs user-defined initialization relating to the data transfer controller.
R_DTCm_Start	Allows starting of the data transfer controller.
R_DTCm_Stop	Prohibits starting of the data transfer controller.

R_DTC_Create

Performs initialization necessary to control the data transfer controller.

[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.

[Syntax]

```
void    R_DTC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTC*m*_Start

Allows starting the data transfer controller.

Remark In this API function, starting the data transfer controller is allowed by operating the DTCE bit of the DTC activation enable register *n* (DTCER*n*) supporting the transfer data number *m*.

[Syntax]

```
void    R_DTCm_Start ( void );
```

Remark *m* is the transfer data number.

[Argument(s)]

None.

[Return value]

None.

R_DTC*m*_Stop

Prohibits starting of the data transfer controller.

Remark In this API function, starting the data transfer controller is prohibited by operating the DTCE bit of the DTC activation enable register *n* (DTCER*n*) supporting the transfer data number *m*.

[Syntax]

```
void    R_DTCm_Stop ( void );
```

Remark *m* is the transfer data number.

[Argument(s)]

None.

[Return value]

None.

C.2.9 Event link controller (ELC)

Below is a list of API functions output by the Code Generator for event link controller use.

Table C-9. API Functions: [Event Link Controller]

API Function Name	Function
R_ELC_Create	Performs initialization necessary to control the event link controller.
R_ELC_Create_UserInit	Performs user-defined initialization relating to the event link controller.
r_elc_elsr18i_interrupt	Performs processing in response to the event link interrupt.
R_ELC_Start	Starts interlinked operation of peripheral functions.
R_ELC_Stop	Ends interlinked operation of peripheral functions.
R_ELC_GenerateSoftwareEvent	Generates the software event.
R_ELC_Set_PortBuffer1	Sets the value of a port buffer.
R_ELC_Get_PortBuffer1	Gets the value of a port buffer.

R_ELC_Create

Performs initialization necessary to control the event link controller.

[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.

[Syntax]

```
void    R_ELC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_elc_elsr18i_interrupt

Performs processing in response to the event link interrupt.

Remark This API function is called to run interrupt processing for the event signal defined in event link setting register 18 (ELSR18).

[Syntax]

```
static  void    r_elc_elsr18i_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_Start

Starts interlinked operation of peripheral functions.

[Syntax]

```
void    R_ELC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_Stop

Ends interlinked operation of peripheral functions.

[Syntax]

```
void    R_ELC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_GenerateSoftwareEvent

Generates the software event.

[Syntax]

```
void    R_ELC_GenerateSoftwareEvent ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_Set_PortBuffer1

Sets the value of a port buffer.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_ELC_Set_PortBuffer1 ( uint8_t value );
```

[Argument(s)]

I/O	Argument	Description
I	`uint8_t value;`	The value set in the port buffer.

[Return value]

None.

R_ELC_Get_PortBuffer1

Gets the value of a port buffer.

[Syntax]

```
#include    "r_cg_macrodriver"
void    R_ELC_Get_PortBuffer1 ( uint8_t * const value );
```

[Argument(s)]

I/O	Argument	Description
O	`uint8_t * const value;`	Pointer to the location where the obtained value is to be stored.

[Return value]

None.

C.2.10 I/O ports

Below is a list of API functions output by the Code Generator for I/O ports use.

Table C-10. API Functions: [I/O Ports]

API Function Name	Function
R_PORT_Create	Performs initialization necessary to control the I/O ports.
R_PORT_Create_UserInit	Performs user-defined initialization relating to the I/O ports.

R_PORT_Create

Performs initialization necessary to control the I/O ports.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_PORT_Create_UserInit

Performs user-defined initialization relating to the I/O ports.

Remark This API function is called as the R_PORT_Create callback routine.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

C.2.11 Multi-function timer pulse unit 2 (MTU2)

Below is a list of API functions output by the Code Generator for multi-function timer pulse unit 2 use.

Table C-11. API Functions: [Multi-Function Timer Pulse Unit 2]

API Function Name	Function
R_MTU2_U0_Create	Performs initialization necessary to control the multi-function timer pulse unit 2.
R_MTU2_U0_Create_UserInit	Performs user-defined initialization relating to the multi-function timer pulse unit 2.
r_mtu2_u0_tgimn_interrupt	Performs processing in response to the input capture/compare match interrupt.
r_mtu2_u0_tcivn_interrupt	Performs processing in response to the overflow interrupt.
r_mtu2_u0_tciun_interrupt	Performs processing in response to the underflow interrupt.
R_MTU2_U0_Cn_Start	Starts counting by a 16-bit timer.
R_MTU2_U0_Cn_Stop	Ends counting by a 16-bit timer.

R_MTU2_U0_Create

Performs initialization necessary to control the multi-function timer pulse unit 2.

[Syntax]

```
void    R_MTU2_U0_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MTU2_U0_Create_UserInit

Performs user-defined initialization relating to the multi-function timer pulse unit 2.

Remark This API function is called as the R_MTU2_U0_Create callback routine.

[Syntax]

```
void    R_MTU2_U0_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu2_u0_tgi*mn*_interrupt

Performs processing in response to the input capture/compare match interrupt.

Remark This API function is called to run interrupt processing for the input capture interrupt generated because multi-function timer pulse unit 2 detected the effective edge of the input signal or for the compare match interrupt generated because the current counter value (value of the timer counter, TCNT) matched the defined counter value (value of the timer general register, TGR).

[Syntax]

```
static  void    r_mtu2_u0_tgimn_interrupt ( void );
```

Remark *m* is the timer general register number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu2_u0_tciv*n*_interrupt

Performs processing in response to the overflow interrupt.

Remark This API function is called to run interrupt processing for the overflow interrupt, which is generated in response to an overflow of the timer counter (TCNT).

[Syntax]

```
static  void    r_mtu2_u0_tcivn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu2_u0_tciu*n*_interrupt

Performs processing in response to the underflow interrupt.

Remark This API function is called to run interrupt processing for the underflow interrupt, which is generated in response to an underflow of the timer counter (TCNT).

[Syntax]

```
static  void    r_mtu2_u0_tciun_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_MTU2_U0_C*n*_Start

Starts counting by the 16-bit timer.

[Syntax]

```
void    R_MTU2_U0_Cn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_MTU2_U0_C*n*_Stop

Ends counting by the 16-bit timer.

[Syntax]

```
void    R_MTU2_U0_Cn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.2.12 Port output enable 2 (POE2)

Below is a list of API functions output by the Code Generator for port output enable 2 use.

Table C-12. API Functions: [Port Output Enable 2]

API Function Name	Function
R_POE2_Create	Performs initialization necessary to control the port output enable 2.
R_POE2_Create_UserInit	Performs user-defined initialization relating to the port output enable 2.
r_poe2_oein_interrupt	Performs processing in response to the output enable interrupt n (OEI*n*).
R_POE2_Start	Places the MTU's complementary PWM output pins in the high-impedance state.
R_POE2_Stop	Releases the R_POE2_Stop MTU's complementary PWM output pins from the high-impedance state.

R_POE2_Create

Performs initialization necessary to control the port output enable 2.

[Syntax]

```
void    R_POE2_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE2_Create_UserInit

Performs user-defined initialization relating to the port output enable 2.

Remark This API function is called as the R_POE2_Create callback routine.

[Syntax]

```
void    R_POE2_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_poe2_oei*n*_interrupt

Performs processing in response to the output enable interrupt *n* (OEI*n*).

Remark This API function is called to run interrupt processing for the output enable interrupt n (OEI*n*), which is generated when a pin (any of POE0#, POE1#, POE2#, POE3#, and POE8#) becomes high-impedance or the output short flag 1 is set.

[Syntax]

```
static  void    r_poe2_oein_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_POE2_Start

Places the MTU's complementary PWM output pins in the high-impedance state.

[Syntax]

```
void    R_CPOE2_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE2_Stop

Releases the MTU's complementary PWM output pins from the high-impedance state.

[Syntax]

```
void    R_POE2_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.2.13 Compare match timer (CMT)

Below is a list of API functions output by the Code Generator for compare match timer use.

Table C-13. API Functions: [Compare Match Timer]

API Function Name	Function
R_CMTn_Create	Performs initialization necessary to control the compare match timer.
R_CMTn_Create_UserInit	Performs user-defined initialization relating to the compare match timer.
r_cmt_cmin_interrupt	Performs processing in response to the compare match interrupt (CMI*n*).
R_CMTn_Start	Starts counting by a 16-bit timer.
R_CMTn_Stop	Ends counting by a 16-bit timer.

R_CMT*n*_Create

Performs initialization necessary to control the compare match timer.

[Syntax]

```
void    R_CMTn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMT*n*_Create_UserInit

Performs user-defined initialization relating to the compare match timer.

Remark This API function is called as the R_CMTn_Create callback routine.

[Syntax]

```
void    R_CMTn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_cmt_cmi*n*_interrupt

Performs processing in response to the compare match interrupt (CMI*n*).

Remark This API function is called to run interrupt processing for the compare match interrupt (CMI*n*), which is generated because the current counter value (value of the compare match timer counter, CMCR) matched the defined counter value (value of the compare match timer constant register, CMCOR).

[Syntax]

```
static  void    r_cmt_cmin_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMT*n*_Start

Starts counting by a 16-bit timer.

[Syntax]

```
void    R_CMTn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMT*n*_Stop

Ends counting by a 16-bit timer.

[Syntax]

```
void    R_CMTn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.2.14 Realtime clock (RTCA)

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

Table C-14. API Functions: [Realtime Clock]

API Function Name	Function
R_RTC_Create	Performs initialization necessary to control the realtime clock.
R_RTC_Create_UserInit	Performs user-defined initialization relating to the realtime clock.
r_rtc_alm_interrupt	Performs processing in response to the alarm interrupt (ALM).
r_rtc_prd_interrupt	Performs processing in response to the periodic interrupt (PRD).
r_rtc_cup_interrupt	Performs processing in response to the carry interrupt (CUP).
R_RTC_Set_CalendarAlarm	Sets the condition for the alarm interrupt (ALM) and allows detection of ALM (calendar count mode).
R_RTC_Set_BinaryAlarm	Sets the condition for the alarm interrupt (ALM) and allows detection of ALM (binary count mode).
R_RTC_Set_ConstPeriodInterruptOn	Sets the period of the periodic interrupt (PRD) and allows detection of PRD.
R_RTC_Set_ConstPeriodInterruptOff	Prohibits detection of the periodic interrupt (PRD).
R_RTC_Set_CarryInterruptOn	Allows detection of the carry interrupt (CUP).
R_RTC_Set_CarryInterruptOff	Prohibits detection of the carry interrupt (CUP).
R_RTC_Set_RTCOUTOn	Set the RTCOUT output period and starts RTCOUT output.
R_RTC_Set_RTCOUTOff	Ends the RTCOUT output.
R_RTC_Start	Starts counting.
R_RTC_Stop	Ends counting.
R_RTC_Restart	Initializes the counter then starts counting.
R_RTC_Set_CalendarCounterValue	Sets the values of the calendar and time counters.
R_RTC_Get_CalendarCounterValue	Gets the values of the calendar and time counters.
R_RTC_Set_BinaryCounterValue	Sets the value of the binary counter.
R_RTC_Get_BinaryCounterValue	Gets the value of the binary counter.

R_RTC_Create

Performs initialization necessary to control the realtime clock.

[Syntax]

```
void    R_RTC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Create_UserInit

Performs user-defined initialization relating to the realtime clock.

Remark This API function is called as the R_RTC_Create callback routine.

[Syntax]

```
void    R_RTC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_alm_interrupt

Performs processing in response to the alarm interrupt (ALM).

Remark This API function is called to run interrupt processing for the alarm interrupt (ALM), which is generated when the condition specified by R_RTC_Set_CalendarAlarm is satisfied.

[Syntax]

```
static  void    r_rtc_alm_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_prd_interrupt

Performs processing in response to the periodic interrupt (PRD).

Remark This API function is called to run interrupt processing for the periodic interrupt (PRD), which is generated when the period specified by R_RTC_Set_ConstPeriodInterruptOn elapses.

[Syntax]

```
static  void    r_rtc_prd_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_cup_interrupt

Performs processing in response to the carry interrupt (CUP).

Remark This API function is called to run interrupt processing for the carry interrupt (CUP), which is generated on carries from the seconds counter (RSECCNT) or binary counter 0 (BCNT0) or when the 64-Hz counter (R64CNT) is read at the same time as a carry from the 64-Hz counter (R64CNT).

[Syntax]

```
static  void    r_rtc_cup_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_CalendarAlarm

Sets the condition for the alarm interrupt (ALM) and allows detection of ALM (calendar count mode).

[Syntax]

```
#include    "r_cg_rtc.h"
void    R_RTC_Set_CalendarAlarm ( rtc_calendar_alarm_enable_t alarm_enable,
rtc_calendar_alarm_value_t alarm_val );
```

[Argument(s)]

I/O	Argument	Description
I	`rtc_calendar_alarm_enable_t alarm_enable;`	Comparison flags (year, month, date, day-of-the-week, hour, minute, and second). 0x0: Comparison proceeds 0x80: Comparison does not proceed
I	`rtc_calendar_alarm_value_t alarm_val;`	Calendar and time values (year, month, date, day-of-week, time, minute, and second)

Remarks 1. The configuration of the comparison flag structure rtc_calendar_alarm_enable_t is shown below.

```
typedef struct {
    uint8_t sec_enb;    /* Second */
    uint8_t min_enb;    /* Minute */
    uint8_t hr_enb;     /* Time */
    uint8_t day_enb;    /* Date */
    uint8_t wk_enb;     /* Day-of-week */
    uint8_t mon_enb;    /* Month */
    uint8_t yr_enb;     /* Year */
} rtc_calendar_alarm_enable_t;
```

2. The configuration of the calendar and time values rtc_calendar_alarm_value_t is shown below.

```
typedef struct {
    uint8_t rsecar;     /* second */
    uint8_t rminar;     /* Minute */
    uint8_t rhrar;      /* Time */
    uint8_t rdayar;     /* Date */
    uint8_t rwkar;      /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmonar;     /* Month */
    uint16_t ryrar;     /* Year */
} rtc_calendar_alarm_value_t;
```

[Return value]

None.

R_RTC_Set_BinaryAlarm

Sets the condition for the alarm interrupt (ALM) and allows detection of ALM (binary count mode).

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_RTC_Set_BinaryAlarm ( uint32_t alarm_enable, uint32_t alarm_val );
```

[Argument(s)]

I/O	Argument	Description
I	`uint32_t alarm_enable;`	Comparison flag 0x0: Comparison does not proceed 0x1: Comparison proceeds
I	`uint32_t alarm_val;`	The value of the binary counter

[Return value]

None.

R_RTC_Set_ConstPeriodInterruptOn

Sets the period of the periodic interrupt (PRD) and allows detection of the PRD.

[Syntax]

```
#include    "r_cg_rtc.h"
void    R_RTC_Set_ConstPeriodInterruptOn ( rtc_int_period_t period );
```

[Argument(s)]

I/O	Argument	Description
I	`rtc_int_period_t period;`	Period of the periodic interrupt (PRD). PES_2_SEC: 2 seconds PES_1_SEC: 1 second PES_1_2_SEC: 1/2 second PES_1_4_SEC: 1/4 second PES_1_8_SEC: 1/8 second PES_1_16_SEC: 1/16 second PES_1_32_SEC: 1/32 second PES_1_64_SEC: 1/64 second PES_1_128_SEC: 1/128 second PES_1_256_SEC: 1/256 second

[Return value]

None.

R_RTC_Set_ConstPeriodInterruptOff

Prohibits detection of the periodic interrupt (PRD).

[Syntax]

```
void    R_RTC_Set_ConstPeriodInterruptOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_CarryInterruptOn

Allows detection of the carry interrupt (CUP).

[Syntax]

```
void    R_RTC_Set_CarryInterruptOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_CarryInterruptOff

Prohibits detection of the carry interrupt (CUP).

[Syntax]

```
void    R_RTC_Set_CarryInterruptOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTCOUTOn

Sets the RTCOUT output period and starts RTCOUT output.

[Syntax]

```
#include    "r_cg_rtc.h"
void    R_RTC_Set_RTCOUTOn ( rtc_rtcout_period_t rtcout_freq );
```

[Argument(s)]

I/O	Argument	Description
I	`rtc_rtcout_period_t rtcout_freq;`	RTCOUT output period RTCOUT_1HZ: 1Hz RTCOUT_64HZ: 64Hz

[Return value]

None.

R_RTC_Set_RTCOUTOff

Ends RTCOUT output.

[Syntax]

```
void    R_RTC_Set_RTCOUTOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Start

Starts counting.

[Syntax]

```
void    R_RTC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Stop

Ends counting.

[Syntax]

```
void    R_RTC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Restart

Initializes the counter then starts counting.

Remarks 1. When the realtime clock is operating in the calendar counting mode, this API function initializes the counters to the values specified by the argument *counter_write_val*.

2. When the realtime clock is operating in the binary counting mode, this API function ignores the value specified by the argument *counter_write_val* and clears the counter to zero.

[Syntax]

```
#include    "r_cg_rtc.h"
void    R_RTC_Restart ( rtc_calendarcounter_value_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	`rtc_calendarcounter_value_t counter_write_val;`	Initial value (year, month, date, day-of-week, time, minute, and second)

Remark The configuration of the initial value rtc_calendarcounter_value_t is shown below.

```
typedef struct {
    uint8_t rseccnt;    /* second */
    uint8_t rmincnt;    /* Minute */
    uint8_t rhrcnt;     /* Time */
    uint8_t rdaycnt;    /* Date */
    uint8_t rwkcnt;     /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmoncnt;    /* Month */
    uint16_t ryrcnt;    /* Year */
} rtc_calendarcounter_value_t;
```

[Return value]

None.

R_RTC_Set_CalendarCounterValue

Sets the calendar and time values.

[Syntax]

```
#include    "r_cg_rtc.h"
void    R_RTC_Set_CalendarCounterValue ( rtc_calendarcounter_value_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	rtc_calendarcounter_value_t *counter_write_val*;	Calendar and time values (year, month, date, day-of-week, time, minute, and second)

Remark The configuration of the calendar and time values rtc_calendarcounter_value_t is shown below.

```
typedef struct {
    uint8_t rseccnt;    /* second */
    uint8_t rmincnt;    /* Minute */
    uint8_t rhrcnt;     /* Time */
    uint8_t rdaycnt;    /* Date */
    uint8_t rwkcnt;     /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmoncnt;    /* Month */
    uint16_t ryrcnt;    /* Year */
} rtc_calendarcounter_value_t;
```

[Return value]

None.

R_RTC_Get_CalendarCounterValue

Gets the calendar and time values.

[Syntax]

```
#include    "r_cg_rtc.h"
void    R_RTC_Get_CalendarCounterValue ( rtc_calendarcounter_value_t * const counter_read_val
);
```

[Argument(s)]

I/O	Argument	Description
O	`rtc_calendarcounter_value_t * const counter_read_val;`	Pointer to the area where the obtained calendar and time values (year, month, date, day-of-week, time, minute, and second) are to be stored

Remark The configuration of the calendar and time values rtc_calendarcounter_value_t is shown below.

```
typedef struct {
    uint8_t rseccnt;    /* second */
    uint8_t rmincnt;    /* Minute */
    uint8_t rhrcnt;     /* Time */
    uint8_t rdaycnt;    /* Date */
    uint8_t rwkcnt;     /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmoncnt;    /* Month */
    uint16_t ryrcnt;    /* Year */
} rtc_calendarcounter_value_t;
```

[Return value]

None.

R_RTC_Set_BinaryCounterValue

Sets the value of the binary counter.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_RTC_Set_BinaryCounterValue ( uint32_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	`uint32_t  counter_write_val;`	The value of the binary counter

[Return value]

None.

R_RTC_Get_BinaryCounterValue

Gets the value of the binary count.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_RTC_Get_BinaryCounterValue ( uint32_t * const counter_read_val );
```

[Argument(s)]

I/O	Argument	Description
O	`uint32_t * const counter_read_val;`	Pointer to an area where the obtained value of the binary counter is to be stored

[Return value]

None.

C.2.15 Independent watchdog timer (IWDT)

Below is a list of API functions output by the Code Generator for independent watchdog timer use.

Table C-15. API Functions: [Independent Watchdog Timer]

API Function Name	Function
R_IWDT_Create	Performs initialization necessary to control the independent watchdog timer.
R_IWDT_Create_UserInit	Performs user-defined initialization relating to the independent watchdog timer.
r_iwdt_nmi_interrupt	Performs processing in response to the non-maskable interrupt (WUNI).
R_IWDT_Restart	Clears the independent watchdog timer counter and resumes counting.

R_IWDT_Create

Performs initialization necessary to control the independent watchdog timer.

[Syntax]

```
void    R_IWDT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IWDT_Create_UserInit

Performs user-defined initialization relating to the independent watchdog timer.

Remark This API function is called as the R_IWDT_Create callback routine.

[Syntax]

```
void    R_IWDT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_iwdt_nmi_interrupt

Performs processing in response to the non-maskable interrupt (WUNI).

Remark This API function is called to run interrupt processing for the non-maskable interrupt (WUNI), which is generated when the down-counter underflows or refreshing proceeds outside the period where it is permitted.

[Syntax]

```
static  void    r_iwdt_nmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IWDT_Restart

Clears the independent watchdog timer counter and resumes counting.

[Syntax]

```
void    R_IWDT_Restart ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.2.16 Serial communications interface (SCI)

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

Table C-16. API Functions: [Serial Communications Interface]

API Function Name	Function
R_SCIn_Create	Performs initialization necessary to control the serial communications interface.
R_SCIn_Create_UserInit	Performs user-defined initialization related to the serial communications interface.
r_scin_transmitend_interrupt	Performs processing in response to the transmit-end interrupts.
r_scin_transmit_interrupt	Performs processing in response to the transmit-data-empty interrupts.
r_scin_receiveend_interrupt	Performs processing in response to the receive-data-full interrupts.
r_scin_receiveerror_interrupt	Performs processing in response to the receive error interrupts.
R_SCIn_Start	Starts SCI communication.
R_SCIn_Stop	Ends SCI communication.
R_SCIn_Serial_Send	Starts SCI transmission (synchronous mode).
R_SCIn_Serial_Receive	Starts SCI reception (synchronous mode).
R_SCIn_Serial_Multiprocessor_Send	Starts SCI transmission (multi-processor communications function).
R_SCIn_Serial_Multiprocessor_Receive	Starts SCI reception (multi-processor communications function).
R_SCIn_Serial_Send_Receive	Starts SCI transmission/reception (clock synchronous mode).
R_SCIn_SmartCard_Send	Starts SCI transmission (smart card interface mode).
R_SCIn_SmartCard_Receive	Starts SCI reception (smart card interface mode).
R_SCIn_IIC_Master_Send	Starts SCI master transmission (simple I^2C mode).
R_SCIn_IIC_Master_Receive	Starts SCI master reception (simple I^2C mode).
R_SCIn_SPI_Master_Send	Starts SCI master transmission (simple SPI mode).
R_SCIn_SPI_Master_Send_Receive	Starts SCI master transmission/reception (simple SPI mode).
R_SCIn_SPI_Slave_Send	Starts SCI slave transmission (simple SPI mode).
R_SCIn_SPI_Slave_Send_Receive	Starts SCI slave transmission/reception (simple SPI mode).
R_SCIn_IIC_StartCondition	Sends the start bit.
R_SCIn_IIC_StopCondition	Sends the stop bit.
r_scin_callback_transmitend	Performs processing in response to the transmit-end interrupts.
r_scin_callback_receiveend	Performs processing in response to the receive-data-full interrupts.
r_scin_callback_receiveerror	Performs processing in response to the receive error interrupts.

R_SCI*n*_Create

Performs initialization necessary to control the serial communication interface.

[Syntax]

```
void    R_SCIn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_Create_UserInit

Performs user-defined initialization related to the serial communications interface.

Remark This API function is called as the R_SCIn_Create callback routine.

[Syntax]

```
void    R_SCIn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_transmitend_interrupt

Performs processing in response to the transmit-end interrupts.

Remark This API function is called to run interrupt processing for the transmit-end interrupts.

[Syntax]

```
static  void    r_scin_transmitend_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_transmit_interrupt

Performs processing in response to the transmit-data-empty interrupts.

Remark This API function is called to run interrupt processing for the transmit-data-empty interrupts.

[Syntax]

```
static  void    r_sci*n*_transmit_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_receiveend_interrupt

Performs processing in response to the receive-data-full interrupts.

Remark This function is called to run interrupt processing for the receive-data-full interrupts.

[Syntax]

```
static  void    r_scin_receiveend_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_receiveerror_interrupt

Performs processing in response to the receive error interrupts.

Remark This API function is called to run interrupt processing for the receive error interrupts.

[Syntax]

```
static  void    r_sci n_receiveerror_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_Start

Starts SCI communication.

[Syntax]

```
void    R_SCIn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_Stop

Ends SCI communication.

[Syntax]

```
void    R_SCIn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_Serial_Send

Starts SCI transmission (asynchronous mode).

Remarks 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

2. When performing a SCI transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_Serial_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 tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_Serial_Receive

Starts SCI reception (asynchronous mode).

Remarks 1. This API function repeats SCI reception in byte units the number of times specified by the argument *rx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

2. When performing a SCI reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_Serial_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 rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *rx_num* specification

R_SCI*n*_Serial_Multiprocessor_Send

Starts SCI transmission (multi-processor communications function).

Remarks 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

2. When performing a SCI transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"

MD_STATUS   R_SCIn_Serial_Multiprocessor_Send ( uint8_t * id_buf, uint16_t id_num, uint8_t *
const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t *` *`id_buf`*`;`	Pointer to a buffer storing the transmission ID
I	`uint16_t` *`id_num`*`;`	Total amount of ID to send
I	`uint8_t * const` *`tx_buf`*`;`	Pointer to a buffer storing the transmission data
I	`uint16_t` *`tx_num`*`;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_Serial_Multiprocessor_Receive

Starts SCI reception (multi-processor communications function).

Remarks 1. This API function repeats SCI reception in byte units the number of times specified by the argument *rx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

2. When performing a SCI reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_Serial_Multiprocessor_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 rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *rx_num* specification

R_SCI*n*_Serial_Send_Receive

Starts SCI transmission/reception (clock synchronous mode).

Remarks 1. This API function repeats SCI transmission in byte units the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

2. This API function repeats SCI reception processing in byte units the number of times specified by the argument *rx_num* and then stores the received data in the buffer at the location specified by the argument r*x_buf*.

3. When performing a SCI transmission/reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SmartCard_Send

Starts SCI transmission (smart card interface mode).

Remarks 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

2. When performing a SCI transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SmartCard_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 tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SmartCard_Receive

Starts SCI reception (smart card interface mode).

Remarks 1. This API function repeats SCI reception in byte units the number of times specified by the argument *rx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

2. When performing a SCI reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SmartCard_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 rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *rx_num* specification

R_SCI*n*_IIC_Master_Send

Starts SCI master transmission (simple I^2C mode).

Remarks 1. This API function handles SCI master transmission to the slave device at the address specified by the argument *adr* and the R/W#bit. SCI master transmission in byte units is repeated the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

2. This API function internally calls R_SCIn_IIC_StartCondition to handle processing to start SCI master transmission.

3. When performing a SCI master transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

R_SCI*n*_IIC_Master_Receive

Starts SCI master reception (simple I^2C mode).

Remarks 1. This API function handles SCI master transmission to the slave device at the address specified by the argument *adr*. SCI master reception in byte units is repeated the number of times specified by the argument *rx_num* and the received data are stored in the buffer at the location specified by the argument *rx_buf*.

2. This API function internally calls R_SCIn_IIC_StartCondition to handle processing to start SCI master reception.

3. When performing a SCI master reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t adr;`	Slave address
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

None.

R_SCI*n*_SPI_Master_Send

Starts SCI master transmission (simple SPI mode).

Remarks 1. This API function repeats the byte-level SCI master transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

2. When performing a SCI master transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SPI_Master_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 tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SPI_Master_Send_Receive

Starts SCI master transmission/reception (simple SPI mode).

Remarks 1. This API function repeats SCI master transmission in byte units the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

2. This API function repeats SCI master reception in byte units the number of times specified by the argument *rx_num* and the received data are stored in the buffer at the location specified by the argument *rx_buf*.

3. When performing a SCI master transmission/reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"

MD_STATUS   R_SCIn_SPI_Master_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num, uint8_t
* const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SPI_Slave_Send

Starts SCI slave transmission (simple SPI mode).

Remarks 1. This API function repeats the byte-level SCI slave transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

2. When performing a SCI slave transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SPI_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 tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SPI_Slave_Send_Receive

Starts SCI slave transmission/reception (simple SPI mode).

Remarks 1. This API function repeats SCI slave transmission in byte units the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

2. This API function repeats SCI slave reception in byte units the number of times specified by the argument *rx_num* and the received data are stored in the buffer at the location specified by the argument *rx_buf*.

3. When performing a SCI slave transmission/reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"

MD_STATUS   R_SCIn_SPI_Slave_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num, uint8_t *
const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t * const tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_IIC_StartCondition

Sends the start bit.

Remark This API function is called as the internal function of R_SCIn_IIC_Master_Send and R_SCIn_IIC_Master_Receive.

[Syntax]

```
void    R_SCIn_IIC_StartCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_IIC_StopCondition

Sends the stop bit.

[Syntax]

```
void    R_SCIn_IIC_StopCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_callback_transmitend

Performs processing in response to the transmit-end interrupts.

Remark This API function is called as the r_scin_transmitend_interrupt callback routine.

[Syntax]

```
static  void    r_sci*n*_callback_transmitend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_callback_receiveend

Performs processing in response to the receive-data-full interrupts.

Remark This API function is called as the r_scin_receiveend_interrupt callback routine.

[Syntax]

```
static  void    r_scin_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_callback_receiveerror

Performs processing in response to the receive error interrupts.

Remark This API function is called as the r_scin_receiveerror_interrupt callback routine.

[Syntax]

```
static  void    r_scin_callback_receiveerror ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.2.17 I²C bus interface (RIIC)

Below is a list of API functions output by the Code Generator for I²C bus interface use.

Table C-17. API Functions: [I²C Bus Interface]

API Function Name	Function
R_RIICn_Create	Performs initialization necessary to control the I²C bus interface.
R_RIICn_Create_UserInit	Performs user-defined initialization relating to the I²C bus interface.
r_riicn_error_interrupt	Performs processing in response to the transfer error/event generation interrupts (EEI).
r_riicn_receive_interrupt	Performs processing in response to the receive data full interrupts (RXI).
r_riicn_transmit_interrupt	Performs processing in response to the transmit data empty interrupts (TXI).
r_riicn_transmitend_interrupt	Performs processing in response to the transmit end interrupts (TEI).
R_RIICn_Start	Starts RIIC communication.
R_RIICn_Stop	Ends RIIC communication.
R_RIICn_Master_Send	Starts RIIC master transmission.
R_RIICn_Master_Receive	Starts RIIC master reception.
R_RIICn_Slave_Send	Starts RIIC slave transmission.
R_RIICn_Slave_Receive	Starts RIIC slave reception.
R_RIICn_StartCondition	Issues the start condition and causes a transfer error and an event generation interrupt (EEI).
R_RIICn_StopCondition	Issues the stop condition and causes a transfer error and an event generation interrupt (EEI).
r_riicn_callback_receiveerror	Of the internal processing for transfer error/event generation interrupts (EEI), this function handles processing specialized in the arbitration-lost detection, NACK detection, and timeout detection.
r_riicn_callback_transmitend	Of the internal processing for transfer error/event generation interrupts (EEI), this function handles processing specialized in the start condition detection in response to calling of R_RIICn_Master_Send.
r_riicn_callback_receiveend	Of the interrupt processing for transfer error/event generation interrupts (EEI), processing specialized in the start condition detection in response to calling of R_RIICn_Master_Receive is performed.

R_RIIC*n*_Create

Performs initialization necessary to control the I^2C bus interface.

[Syntax]

```
void    R_RIICn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Create_UserInit

Performs user-defined initialization relating to the I^2C bus interface.

Remark This API function is called as the R_RIICn_Create callback routine.

[Syntax]

```
void    R_RIICn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_error_interrupt

Performs processing in response to the transfer error/event generation interrupts (EEI).

Remark This API function is called to run interrupt processing for the transfer error/event generation interrupts (EEI), which are generated when the I^2C bus interface detects the transfer error/event generation (arbitration-lost, NACK, timeout, start condition, and stop condition).

[Syntax]

```
static  void    r_riicn_error_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_receive_interrupt

Performs processing in response to the receive data full interrupts (RXI).

Remark This API function is called to run interrupt processing for the receive data full interrupts (RXI).

[Syntax]

```
static  void    r_riicn_receive_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_transmit_interrupt

Performs processing in response to the transmit data empty interrupts (TXI).

Remark This function is called to run interrupt processing for the transmit data empty interrupts (TXI).

[Syntax]

```
static  void    r_riicn_transmit_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_transmitend_interrupt

Performs processing in response to the transmit end interrupts (TEI).

Remark This API function is called to run interrupt processing for the transmit end interrupts (TEI).

[Syntax]

```
static  void    r_riicn_transmitend_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Start

Starts RIIC communication.

[Syntax]

```
void    R_RIICn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Stop

Ends RIIC communication.

[Syntax]

```
void    R_RIICn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Master_Send

Starts RIIC master transmission.

Remarks 1. This API function handles RIIC master transmission to the slave device at the address specified by the argument *adr* and the R/W#bit. RIIC master transmission in byte units is repeated the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

2. This API function internally calls R_RIICn_StartCondition to handle processing to start RIIC master transmission.

3. When performing a RIIC master transmission, R_RIICn_Start must be called before this API function is called.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus busy
MD_ERROR2	Invalid argument *adr* specification

R_RIIC*n*_Master_Receive

Starts RIIC master reception.

Remarks 1. This API function handles RIIC master transmission to the slave device at the slave address specified by the argument *adr*. RIIC master reception in byte units is repeated the number of times specified by the argument *rx_num* and the received data are stored in the buffer at the location specified by the argument *rx_buf*.

2. This API function internally calls R_RIICn_StartCondition to handle processing to start RIIC master reception.

3. When performing a RIIC master reception, R_RIICn_Start must be called before this API function is called.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t adr;`	Slave address
O	`uint8_t * const rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus busy
MD_ERROR2	Invalid argument *adr* specification

R_RIIC*n*_Slave_Send

Starts RIIC slave transmission.

Remarks 1. This API function repeats the byte-level RIIC slave transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

2. When performing a RIIC slave transmission, R_RIICn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_RIICn_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 tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send

[Return value]

None.

R_RIIC*n*_Slave_Receive

Starts RIIC slave reception.

Remarks 1. This API function performs byte-level RIIC 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*.

2. When performing a RIIC slave reception, R_RIICn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_RIICn_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 rx_buf;`	Pointer to a buffer to store the reception data
I	`uint16_t rx_num;`	Total amount of data to receive

[Return value]

None.

R_RIIC*n*_StartCondition

Issues the start condition and causes a transfer error and an event generation interrupt (EEI).

Remarks 1. This API function is called as the internal function of R_RIICn_Master_Send and R_RIICn_Master_Receive.

2. r_riicn_error_interrupt is called in response to calling of this API function.

[Syntax]

```
void    R_RIICn_StartCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_StopCondition

Issues the stop condition and causes a transfer error and an event generation interrupt (EEI).

Remark r_riicn_error_interrupt is called in response to calling of this API function.

[Syntax]

```
void    R_RIICn_StopCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_callback_receiveerror

Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the arbitration-lost detection, NACK detection, and timeout detection.

Remark This API function is called as the r_riicn_error_interrupt callback routine.

[Syntax]

```
#include    "r_cg_macrodriver.h"
static  void    r_riicn_callback_receiveerror ( MD_STATUS status );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`MD_STATUS status;`	Source of the transfer errors and event generation interrupts MD_ERROR1: Arbitration-lost detection MD_ERROR2: Timeout detection MD_ERROR3: NACK detection

[Return value]

None.

r_riic*n*_callback_transmitend

Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the start condition detection in response to calling of R_RIICn_Master_Send.

Remark This API function is called as the r_riicn_error_interrupt callback routine.

[Syntax]

```
static  void    r_riicn_callback_transmitend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_callback_receiveend

Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the start condition detection in response to calling of R_RIICn_Master_Receive.

Remark This API function is called as the r_riicn_error_interrupt callback routine.

[Syntax]

```
static  void    r_riicn_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.2.18 Serial peripheral interface (RSPI)

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

Table C-18. API Functions: [Serial Peripheral Interface]

API Function Name	Function
R_RSPIn_Create	Performs initialization necessary to control the serial peripheral interface.
R_RSPIn_Create_UserInit	Performs user-defined initialization relating to the serial peripheral interface.
r_rspin_receive_interrupt	Performs processing in response to the receive buffer full interrupts.
r_rspin_transmit_interrupt	Performs processing in response to the transmit buffer error interrupts.
r_rspin_error_interrupt	Performs processing in response to the RSPI error interrupts.
r_rspin_idle_interrupt	Performs processing in response to the RSPI idle interrupts.
R_RSPIn_Start	Starts RSPI communication.
R_RSPIn_Stop	Ends RSPI communication.
R_RSPIn_LoopBackReversed	Enables the loopback mode (reversed transmit data = receive data).
R_RSPIn_LoopBackDirect	Enables the loopback mode (transmit data = receive data).
R_RSPIn_LoopBackDisable	Disable the loopback mode and returns the interface to the normal mode.
R_RSPIn_Send	Starts RSPI transmission.
R_RSPIn_Send_Receive	Starts RSPI transmission/reception.
r_rspin_callback_receiveend	Performs processing in response to the receive buffer full interrupts.
r_rspin_callback_error	Performs processing in response to the RSPI error interrupts.
r_rspin_callback_transmitend	Performs processing in response to the RSPI idle interrupts.

R_RSPI*n*_Create

Performs initialization necessary to control the serial peripheral interface.

[Syntax]

```
void    R_RSPIn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Create_UserInit

Performs user-defined initialization relating to the serial peripheral interface.

Remark This API function is called as the R_RSPIn_Create callback routine.

[Syntax]

```
void    R_RSPIn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_receive_interrupt

Performs processing in response to the receive buffer full interrupts.

Remark This API function is called to run interrupt processing for the receive buffer full interrupt.

[Syntax]

```
static  void    r_rspin_receive_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_transmit_interrupt

Performs processing in response to the transmit buffer empty interrupts.

Remark This API function is called to run interrupt processing for the transmit buffer empty interrupts.

[Syntax]

```
static  void    r_rspin_transmit_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_error_interrupt

Performs processing in response to the RSPI error interrupts.

Remark This API function is called to run interrupt processing for the RSPI error interrupts.

[Syntax]

```
static  void    r_rspin_error_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_idle_interrupt

Performs processing in response to the RSPI idle interrupts.

Remark This API function is called to run interrupt processing for the RSPI idle interrupts.

[Syntax]

```
static  void    r_rspin_idle_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Start

Starts RSPI communication.

[Syntax]

```
void    R_RSPIn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Stop

Ends RSPI communication.

[Syntax]

```
void    R_RSPIn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_LoopBackReversed

Enables the loopback mode (reserved transmit data = receive data).

[Syntax]

```
void    R_RSPIn_LoopBackReversed ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_LoopBackDirect

Enables the loopback mode (transmit data = receive data).

[Syntax]

```
void    R_RSPIn_LoopBackDirect ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_LoopBackDisable

Disables the loopback mode and returns the interface to the normal mode.

[Syntax]

```
void    R_RSPIn_LoopBackDisable ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Send

Starts RSPI transmission.

Remarks 1. This API function repeats the byte-level RSPI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

2. When performing a RSPI transmission, R_RSPIn_Start must be called before this API function is called.

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_RSPIn_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 tx_buf;`	Pointer to a buffer storing the transmission data
I	`uint16_t tx_num;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_RSPI*n*_Send_Receive

Starts RSPI transmission/reception.

Remarks 1. This API function repeats RSPI transmission in byte units the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

2. This API function repeats RSPI reception processing in byte units the number of times specified by the argument *tx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

3. When performing a RSPI transmission/reception, R_RSPIn_Start must be called before this API function is called.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

r_rspi*n*_callback_receiveend

Performs processing in response to the receive buffer full interrupts.

Remark This API function is called as the r_rspin_receive_interrupt callback routine.

[Syntax]

```
static  void    r_rspin_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_callback_error

Performs processing in response to the RSPI error interrupts.

Remark This API function is called as the r_rspin_error_interrupt callback routine.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t err_type;`	Source of the RSPI error interrupt (x is undefined) xxxx00x1B: Overrun error detection xxxx01x0B: Mode fault error detection xxxx10x0B: Parity error detection

[Return value]

None.

r_rspi*n*_callback_transmitend

Performs processing in response to the RSPI idle interrupts.

Remark This API function is called as the r_rspin_idle_interrupt callback routine.

[Syntax]

```
static  void    r_rspin_callback_transmitend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.2.19 CRC calculator (CRC)

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

Table C-19. API Functions: [CRC calculator]

API Function Name	Function
R_CRC_SetCRC8	Initializes the CRC calculator for 8-bit CRC calculation (CRC generating polynomial: $X^8 + X^2 + X + 1$).
R_CRC_SetCRC16	Initializes the CRC calculator for 16-bit CRC calculation (CRC generating polynomial: $X^{16} + X^{15} + X^2 + 1$).
R_CRC_SetCCITT	Initializes the CRC calculator for 16-bit CRC calculation (CRC generating polynomial: $X^{16} + X^{12} + X^5 + 1$).
R_CRC_Input_Data	Sets the initial value of the data from which the CRC is to be calculated.
R_CRC_Get_Result	Gets the result of operation.

R_CRC_SetCRC8

Initializes the CRC calculator for 8-bit CRC calculation (CRC generating polynomial: $X^8 + X^2 + X + 1$).

[Syntax]

```
#include    "r_cg_crc.h"
void    R_CRC_SetCRC8 ( crc_bitorder order );
```

[Argument(s)]

I/O	Argument	Description
I	`crc_bitorder order;`	CRC calculation switching type CRC_LSB: LSB-first CRC_MSB: MSB-first

[Return value]

None.

R_CRC_SetCRC16

Initializes the CRC calculator for 16-bit CRC calculation (CRC generating polynomial: $X^{16} + X^{15} + X^2 + 1$).

[Syntax]

```
#include    "r_cg_crc.h"
void    R_CRC_SetCRC16 ( crc_bitorder order );
```

[Argument(s)]

I/O	Argument	Description
I	`crc_bitorder order;`	CRC calculation switching type CRC_LSB: LSB-first CRC_MSB: MSB-first

[Return value]

None.

R_CRC_SetCCITT

Initializes the CRC calculator for the 16-bit CRC calculation (CRC generating polynomial: $X^{16} + X^{12} + X^5 + 1$).

[Syntax]

```
#include    "r_cg_crc.h"
void    R_CRC_SetCCITT ( crc_bitorder order );
```

[Argument(s)]

I/O	Argument	Description
I	`crc_bitorder order;`	CRC calculation switching type CRC_LSB: LSB-first CRC_MSB: MSB-first

[Return value]

None.

R_CRC_Input_Data

Sets the initial value of the data from which the CRC is to be calculated.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_CRC_Input_Data ( uint8_t data );
```

[Argument(s)]

I/O	Argument	Description
I	`uint8_t data;`	The initial value of the data from which the CRC is to be calculated

[Return value]

None.

R_CRC_Get_Result

Gets the result of operation.

[Syntax]

```
#include    "r_cg_macrodriver"
void    R_CRC_Get_Result ( uint8_t * const result );
```

[Argument(s)]

I/O	Argument	Description
O	`uint8_t * const result;`	Pointer to the location where the result of operation is stored

[Return value]

None.

C.2.20 12-bit A/D converter (S12AD)

Below is a list of API functions output by the Code Generator for 12-bit A/D converter use.

Table C-20. API Functions: [12-Bit A/D Converter]

API Function Name	Function
R_S12ADb_Create	Performs initialization necessary to control the 12-bit A/D converter.
R_S12ADb_Create_UserInit	Performs user-defined initialization relating to the 12-bit A/D converter.
r_s12ad_interrupt	Performs processing in response to the A/D scan end interrupt.
r_s12ad_groupb_interrupt	Performs processing in response to the group B scan end interrupt.
R_S12ADb_Start	Starts A/D conversion.
R_S12ADb_Stop	Ends A/D conversion.
R_S12ADb_Get_ValueResult	Gets the result of conversion.

R_S12ADb_Create

Performs initialization necessary to control the 12-bit A/D converter.

[Syntax]

```
void    R_S12ADb_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_S12ADb_Create_UserInit

Performs user-defined initialization relating to the 12-bit A/D converter.

Remark This API function is called as the R_S12ADb_Create callback routine.

[Syntax]

```
void    R_S12ADb_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_s12ad_interrupt

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

Remark This API function is called to run interrupt processing for the A/D scan end interrupt, which is generated on completion of scanning of the analog inputs.

[Syntax]

```
static  void    r_s12ad_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_s12ad_groupb_interrupt

Performs processing in response to the group B scan end interrupt.

Remark This function is called to run interrupt processing for the group B scan end interrupt, which is generated when scanning of the analog inputs allocated to group B is completed.

[Syntax]

```
static  void    r_s12ad_groupb_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_S12ADb_Start

Starts A/D conversion.

[Syntax]

```
void    R_S12ADb_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_S12ADb_Stop

Ends A/D conversion.

[Syntax]

```
void    R_S12ADb_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_S12ADb_Get_ValueResult

Gets the result of conversion.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_s12adb.h"
void    R_S12ADb_Get_ValueResult ( ad_channel_t channel, uint16_t * const buffer );
```

[Argument(s)]

I/O	Argument	Description
I	`ad_channel_t channel;`	Channel number ADCHANNEL0: Input channel AN000 ADCHANNEL1: Input channel AN001 ADCHANNEL2: Input channel AN002 ADCHANNEL3: Input channel AN003 ADCHANNEL4: Input channel AN004 ADCHANNEL6: Input channel AN006 ADCHANNEL8: Input channel AN008 ADCHANNEL9: Input channel AN009 ADCHANNEL10: Input channel AN010 ADCHANNEL11: Input channel AN011 ADCHANNEL12: Input channel AN012 ADCHANNEL13: Input channel AN013 ADCHANNEL14: Input channel AN014 ADCHANNEL15: Input channel AN015 ADTEMPSENSOR: Extended analog input (temperature sensor output) ADINTERREFVOLT: Extended analog input (internal reference voltage)
O	`uint16_t * const buffer;`	Pointer to the area where the results of conversion are stored

[Return value]

None.

C.2.21 D/A converter (DA)

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

Table C-21. API Functions: [D/A Converter]

API Function Name	Function
R_DAC_Create	Performs initialization necessary to control the D/A converter.
R_DAC_Create_UserInit	Performs user-defined initialization relating to the D/A converter.
R_DACm_Start	Starts D/A conversion.
R_DACm_Stop	Ends D/A conversion.
R_DACm_Set_ConversionValue	Sets the data for D/A conversion.

R_DAC_Create

Performs initialization necessary to control the D/A converter.

[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.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_DAC*m*_Start

Starts D/A conversion.

[Syntax]

```
void    R_DACm_Start ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DAC*m*_Stop

Ends D/A conversion.

[Syntax]

```
void    R_DACm_Stop ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DAC*m*_Set_ConversionValue

Sets the data for D/A conversion.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_DACm_Set_ConversionValue ( uint16_t reg_value );
```

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint16_t reg_value;`	Data for D/A conversion

[Return value]

None.

C.2.22 Data operation circuit (DOC)

Below is a list of API functions output by the Code Generator for data operation circuit.

Table C-22. API Functions: [Data Operation Circuit]

API Function Name	Function
R_DOC_Create	Performs initialization necessary to control the data operation circuit.
R_DOC_Create_UserInit	Performs user-defined initialization relating to the data operation circuit.
r_doc_dopcf_interrupt	Performs processing in response to the data operation circuit interrupt.
R_DOC_SetMode	Sets the operating mode and the initial value of the reference value for use by the data operation circuit.
R_DOC_WriteData	Sets the input value (value for comparison with, addition to, or subtraction from the reference value) for use by the data operation circuit.
R_DOC_GetResult	Gets the result of operation.
R_DOC_ClearFlag	Clears the data operation circuit flag.

R_DOC_Create

Performs initialization necessary to control the data operation circuit.

[Syntax]

```
void    R_DOC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DOC_Create_UserInit

Performs user-defined initialization relating to the data operation circuit.

Remark This API function is called as the R_DOC_Create callback routine.

[Syntax]

```
void    R_DOC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_doc_dopcf_interrupt

Performs processing in response to the data operation circuit interrupt.

Remark This API function is called to run interrupt processing for the data operation circuit interrupt, which is generated when the result of data comparison satisfies the condition for detection, the result of addition is greater than 0xFFFF, or the result of subtraction is less than 0x00.

[Syntax]

```
static  void    r_doc_dopcf_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DOC_SetMode

Sets the operating mode and the initial value of the reference value for use by the data operation circuit.

Remarks 1. When COMPARE_MISMATCH or COMPARE_MATCH (data comparison mode) is specified as the mode of operation, the 16-bit reference value is stored in the DOC data setting register (DODSR).

2. When ADDITION (data addition mode) or SUBTRACTION (data subtraction mode) is specified for the mode (operation mode), the 16-bit value is stored in the DOC data setting register (DODSR) as the initial value.

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_doc.h"
void    R_DOC_SetMode ( doc_mode_t mode, uint16_t value );
```

[Argument(s)]

I/O	Argument	Description
I	`doc_mode_t mode;`	Operating modes (including the condition for detection) COMPARE_MISMATCH: Data comparison mode (mismatch) COMPARE_MATCH: Data comparison mode (match) ADDITION: Data addition mode SUBTRACTION: Data subtraction mode
I	`uint16_t value;`	Initial value of the reference value for use by the DOC

[Return value]

None.

R_DOC_WriteData

Sets the value for comparison with, addition to, or subtraction from the reference value.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_DOC_WriteData ( uint16_t data );
```

[Argument(s)]

I/O	Argument	Description
I	`uint16_t  data;`	Input data for use in operation

[Return value]

None.

R_DOC_GetResult

Gets the result of operation.

[Syntax]

```
#include    "r_cg_macrodriver"
void    R_DOC_GetResult ( uint16_t * const data );
```

[Argument(s)]

I/O	Argument	Description
O	`uint16_t * const data;`	Pointer to the location where the result of operation is to be stored

[Return value]

None.

R_DOC_ClearFlag

Clears the data operation circuit flag.

[Syntax]

```
void    R_DOC_ClearFlag ( void );
```

[Argument(s)]

None.

[Return value]

None.

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep 01, 2013	-	First Edition issued

CubeSuite+ V2.01.00 User's Manual:
RX Design

Publication Date: Rev.1.00 Sep 01, 2013

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

Colophon 1.3

CubeSuite+ V2.01.00

Renesas Electronics Corporation

R20UT2683EJ0100