User's Manual

CubeSuite+ V1.00.01

Integrated Development Environment
User's Manual: RL78 Design

Target Device
RL78 Family

Renesas Electronics
www.renesas.com

Rev.1.00 Jun 2011

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. 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.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. 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.
5. When exporting the 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. You should not use Renesas Electronics products or the 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. 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.
6. 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.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. 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 categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. 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 an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "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.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. 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.
9. 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 system manufactured by you.
10. 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.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

How to Use This Manual

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

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

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

Readers

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

Purpose

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

Organization

This manual can be broadly divided into the following units.

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

How to Read This Manual

It is assumed that the readers of this manual have general knowledge of electricity, logic circuits, and microcontrollers.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active low representation:	$\overline{XXX}$ (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	R20UT0545E
	78K0 Design	R20UT0546E
	78K0R Design	R20UT0547E
	RL78 Design	This manual
	V850 Design	R20UT0549E
	R8C Design	R20UT0550E
	78K0 Coding	R20UT0551E
	RL78,78K0R Coding	R20UT0552E
	V850 Coding	R20UT0553E
	Coding for CX Compiler	R20UT0554E
	R8C Coding	R20UT0576E
	78K0 Build	R20UT0555E
	RL78,78K0R Build	R20UT0556E
	V850 Build	R20UT0557E
	Build for CX Compiler	R20UT0558E
	R8C Build	R20UT0575E
	78K0 Debug	R20UT0559E
	78K0R Debug	R20UT0560E
	RL78 Debug	R20UT0548E
	V850 Debug	R20UT0562E
	R8C Debug	R20UT0574E
	Analysis	R20UT0563E
	Message	R20UT0407E

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

[MEMO]

[MEMO]

[MEMO]

TABLE OF CONTENTS

CHAPTER 1 GENERAL ... 10

1.1 Overview ... 10
1.2 Features ... 10

CHAPTER 2 FUNCTIONS (Code Generator) ... 11

2.1 Overview ... 11
2.2 Open Code Generator Panel ... 12
2.3 Enter Information ... 13
2.3.1 Input rule ... 13
2.3.2 Icon indicating incorrect entry ... 14
2.3.3 Icon indicating pin conflict ... 15
2.4 Confirm Source Code ... 16
2.5 Output Source Code ... 17
2.5.1 Set whether or not to generate source code ... 18
2.5.2 Change file name ... 19
2.5.3 Change API function name ... 20
2.5.4 Change output mode ... 21
2.5.5 Change output destination folder ... 22
2.6 Output Report Files ... 23
2.6.1 Change output format ... 24
2.6.2 Change output destination ... 26

APPENDIX A WINDOW REFERENCE ... 27

A.1 Description ... 27

APPENDIX B OUTPUT FILES ... 54

B.1 Overview ... 54
B.2 Output File ... 54

APPENDIX C API FUNCTIONS ... 59

C.1 Overview ... 59
C.2 Output Function ... 59
C.3 Function Reference ... 67
C.3.1 Clock Generator ... 69
C.3.2 Port ... 75
C.3.3 Interrupt ... 78
C.3.4 Serial ... 89

C.3.5 A/D Converter ... 145
C.3.6 D/A Converter ... 160
C.3.7 Timer ... 167
C.3.8 Watchdog Timer ... 203
C.3.9 Real-time Clock ... 208
C.3.10 Interval Timer ... 229
C.3.11 Comparator ... 236
C.3.12 Clock Output/Buzzer Output ... 242
C.3.13 Data Transfer Controller ... 247
C.3.14 Event Link Controller ... 253
C.3.15 DMA Controller ... 257
C.3.16 Voltage Detector ... 264

APPENDIX D INDEX ... 269

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 provided by the microcontroller (clock generator, port functions, etc.) 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 and link directive files.
  Source code output by the Code Generator conforms to the MISRA-C (Guidelines for the Use of the C Language in Vehicle Based Software) coding convention.

- 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 (Code Generator)

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

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 peripheral functions provided by the microcontroller (clock generator, port functions, etc.).

The following sections describe the operation procedures for Code Generator.

(1) Start CubeSuite+

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

Remark See "CubeSuite+ Start User's Manual" for details on "Start CubeSuite+".

(2) Create/Open project

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

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

(3) Open Code Generator Panel

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

(4) Enter Information

Configure the information necessary to control the peripheral functions in the Code Generator panel.

(5) Confirm Source Code

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

(6) Output Source Code

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

(7) Output Report Files

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

(8) Save project

Save a project.

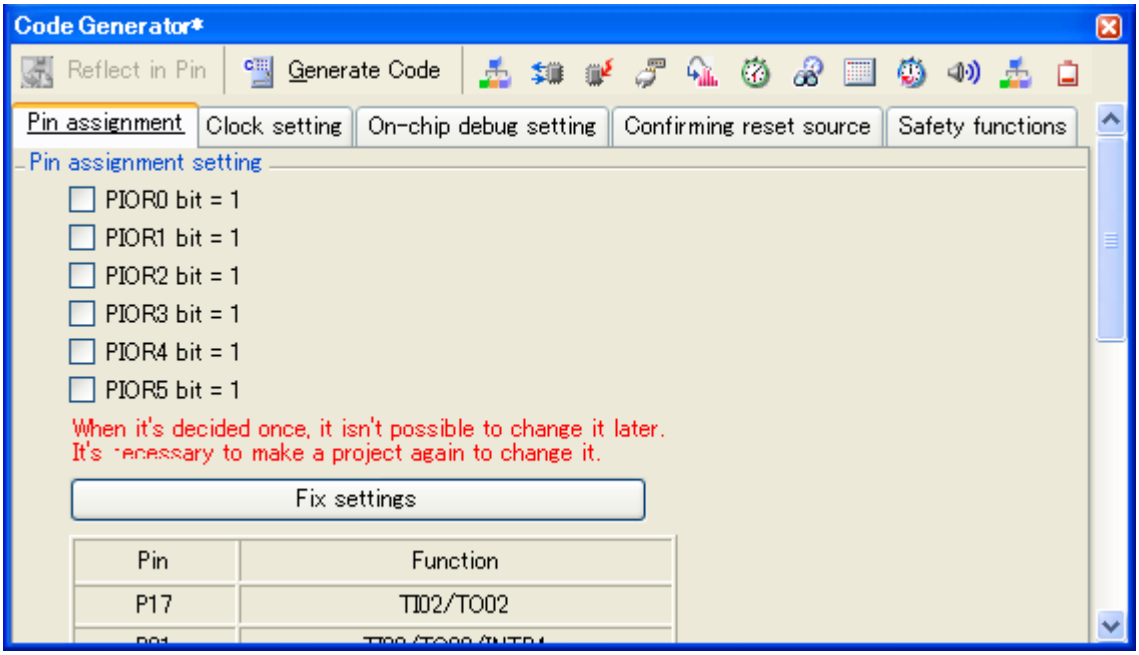
Remark See "CubeSuite+ Start User's Manual" for details on "Save project".

2.2 Open Code Generator Panel

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

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

Figure 2-1. Open Code Generator Panel



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

2.3 Enter Information

Configure the information necessary to control the peripheral functions in the information setting area of the Code Generator panel which is opened as described in "2.2 Open Code Generator Panel".

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

2.3.1 Input rule

Following is the rules for input to the Code Generator panel.

(1) Character set

Character sets that are allowed to input are as follows.

Table 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

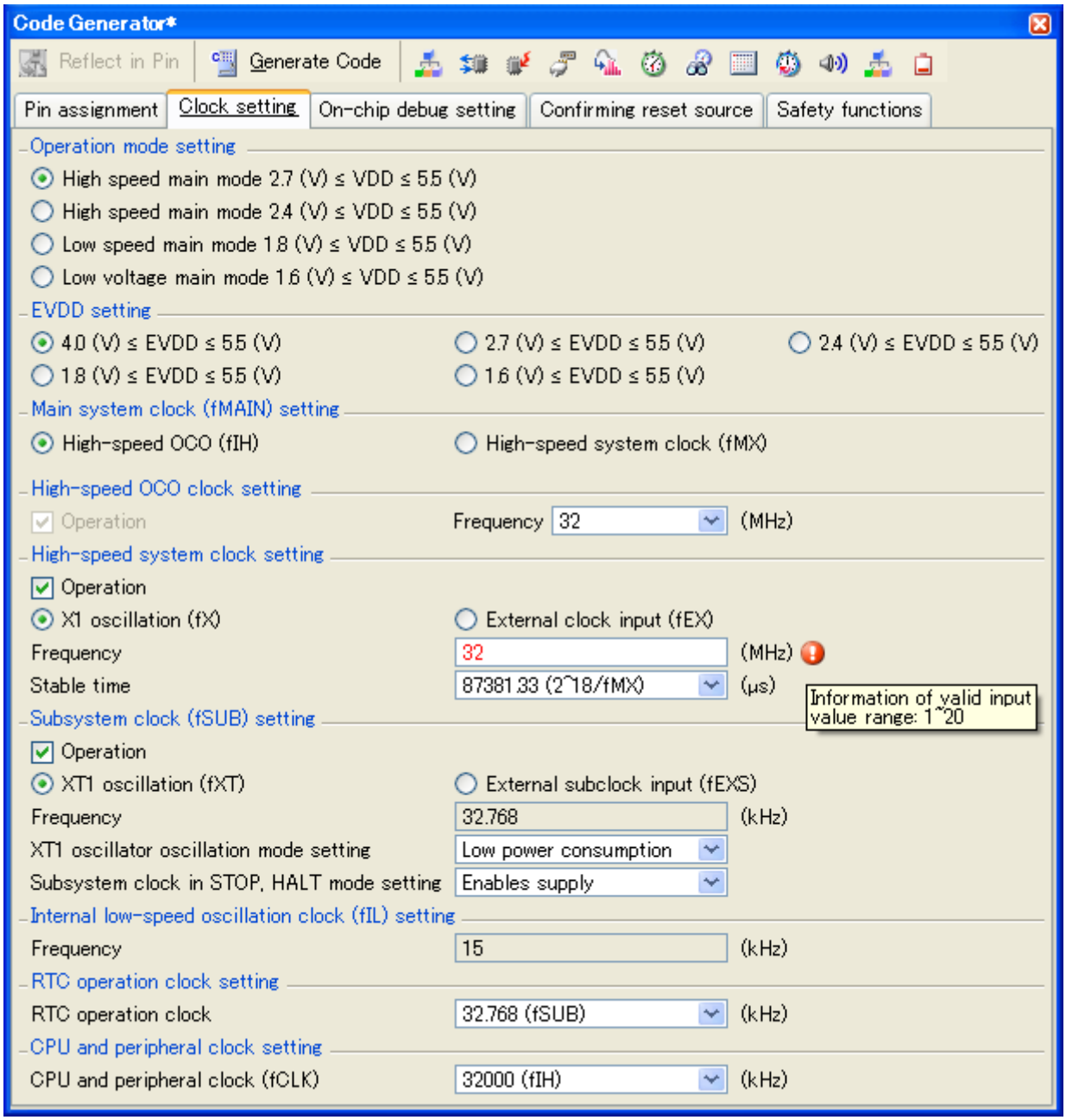
Notation	Outline
Decimal number	A numeric value that starts with a number between 1 and 9 and followed by numbers between 0 and 9, and the numeric value 0
Hex number	A numeric value that starts with 0x and followed by a combination of numbers from 0 to 9 and characters from A to F (characters are not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

2.3.2 Icon indicating incorrect entry

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

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

Figure 2-2. Icon Indicating Incorrect Entry

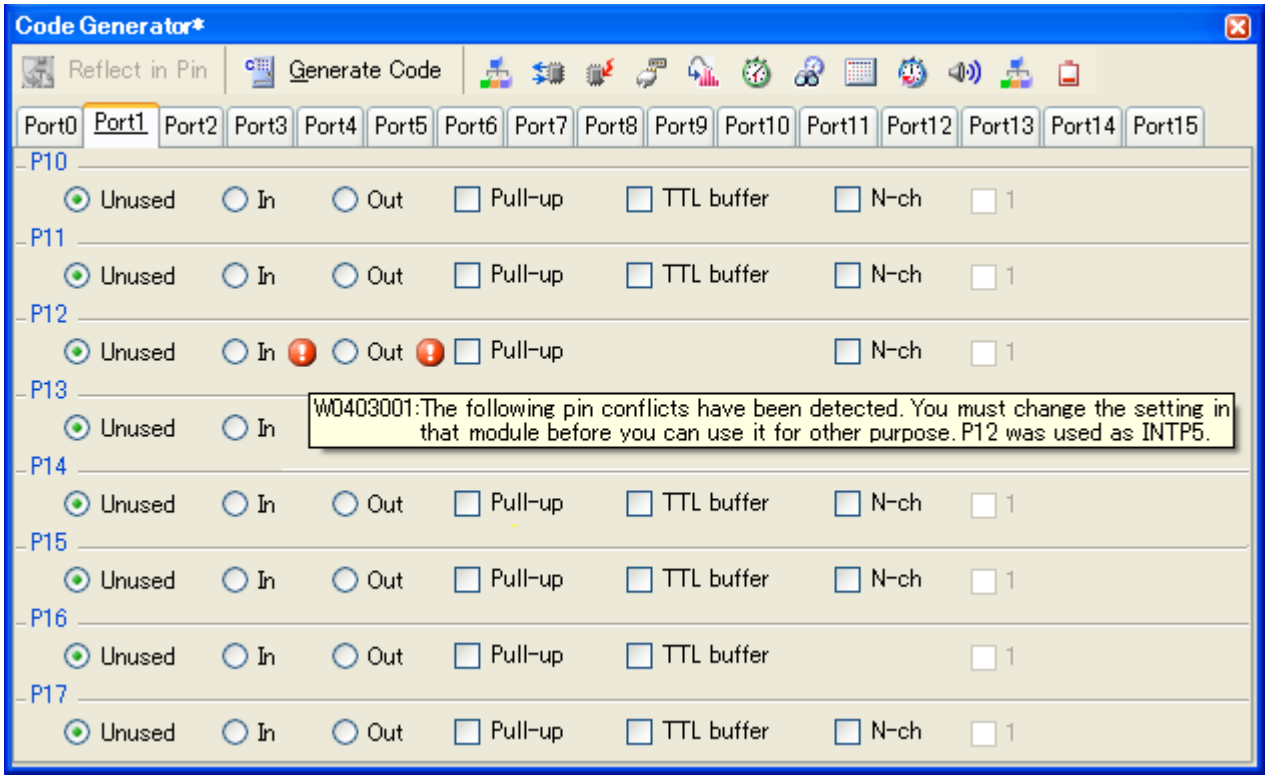


2.3.3 Icon indicating pin conflict

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

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

Figure 2-3. Icon Indicating Pin Conflict

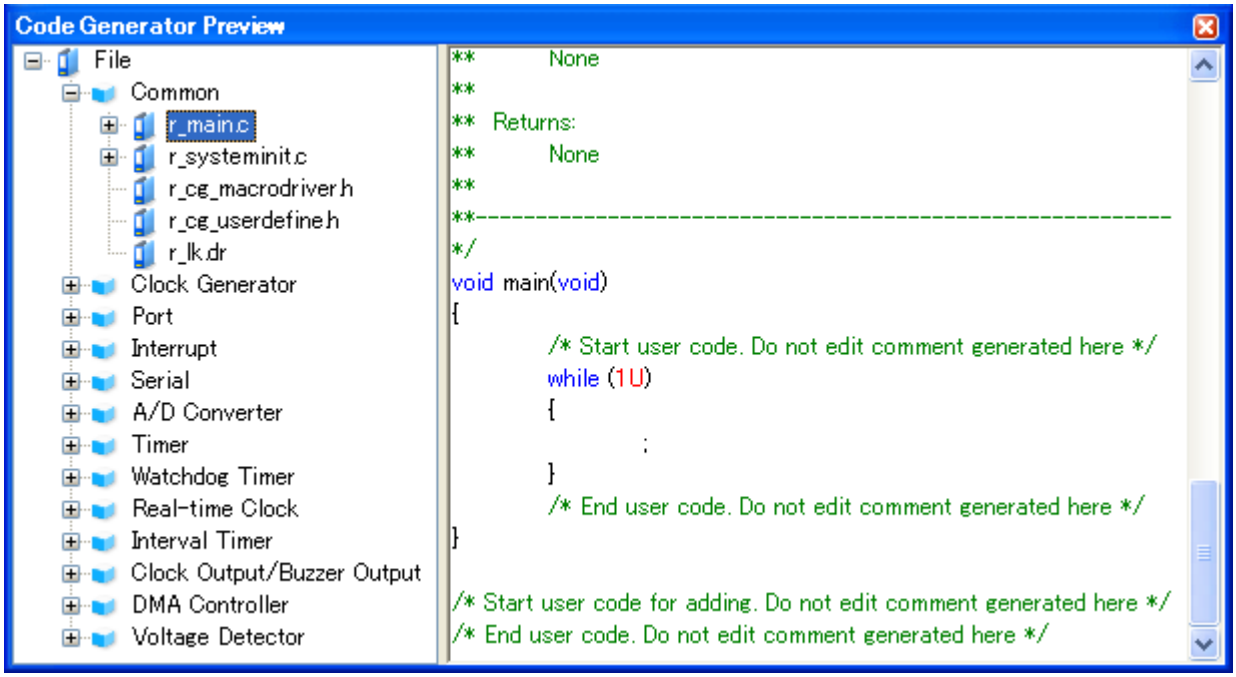


2.4 Confirm Source Code

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 Generator Preview panel that opens by selecting [View] menu >> [Code Generator Preview].

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 Code Generator Preview panel.

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

Table 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 Generator Preview panel.

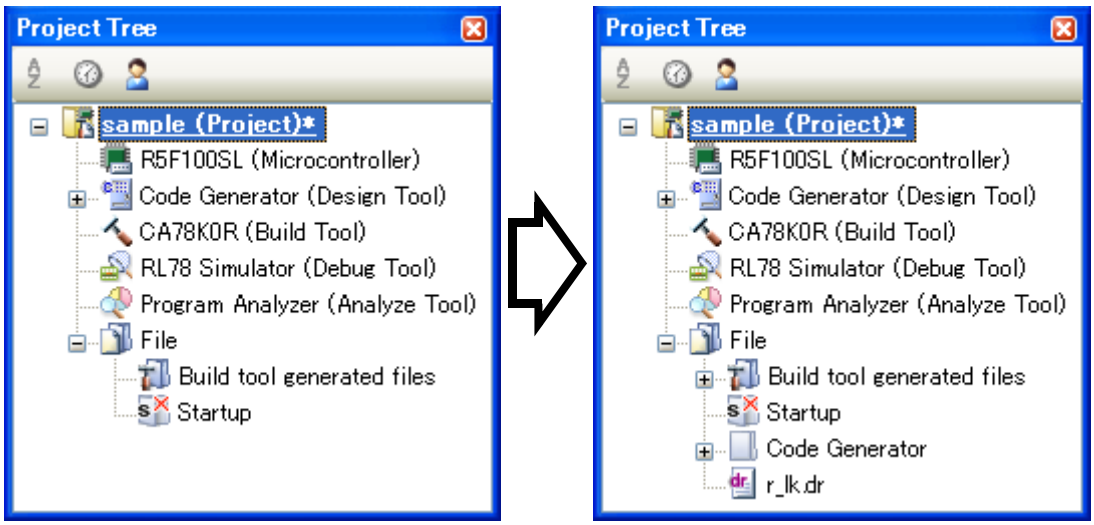
4. For some of the API functions (such as API functions for serial array units), values such as the SFR register value are calculated and finalized when the source code is generated (when the Generate Code button on the Code Generator panel is pressed). For this reason, the source code displayed in the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

2.5 Output Source Code

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

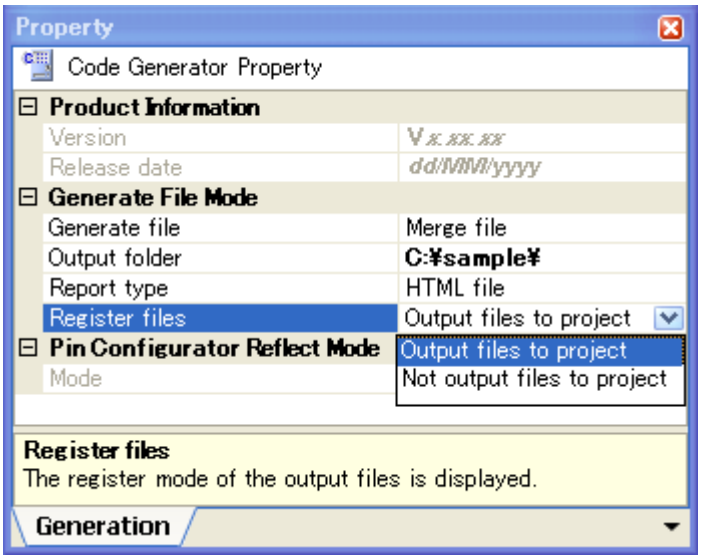
The destination folder for the source code is specified by clicking [Generation] tab >> [Output folder] in the Property panel.

Figure 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 [Generation] tab >> [Register files], specify "Output files to project".

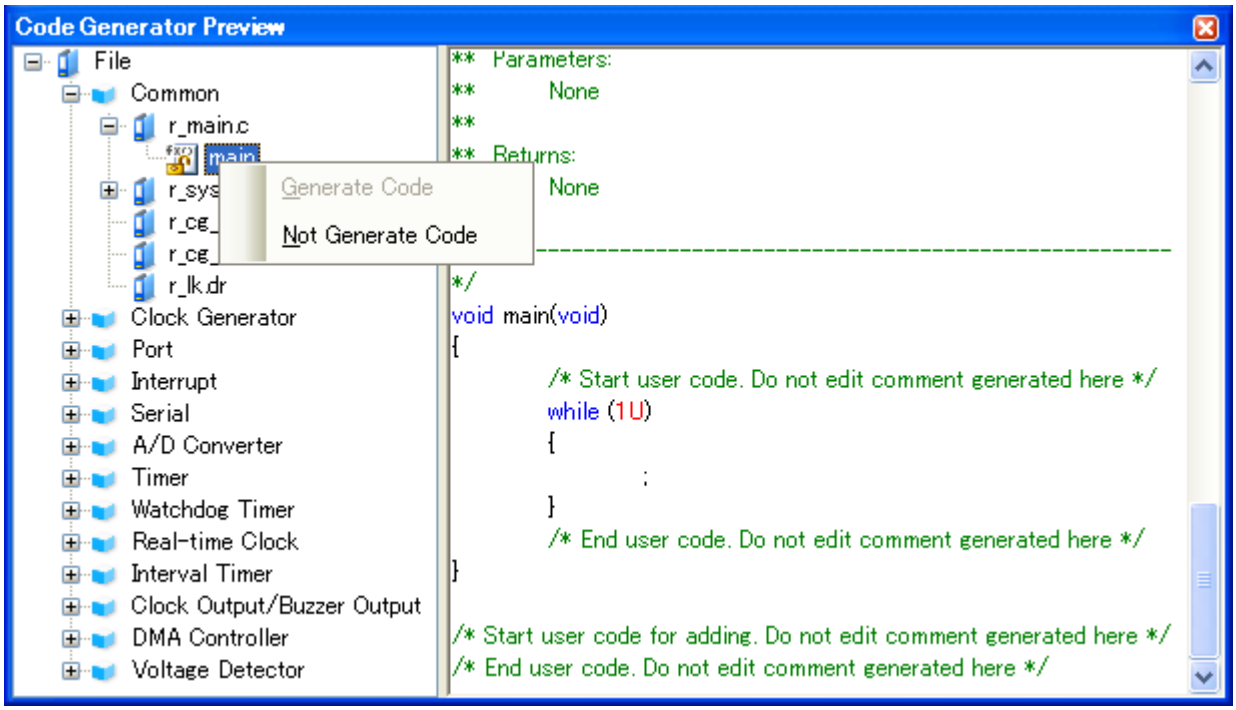
Figure 2-6. Configure Whether to Register



2.5.1 Set whether or not to generate source code

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

Figure 2-7. Setting That Determines Whether or Not to Generate Source Code



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

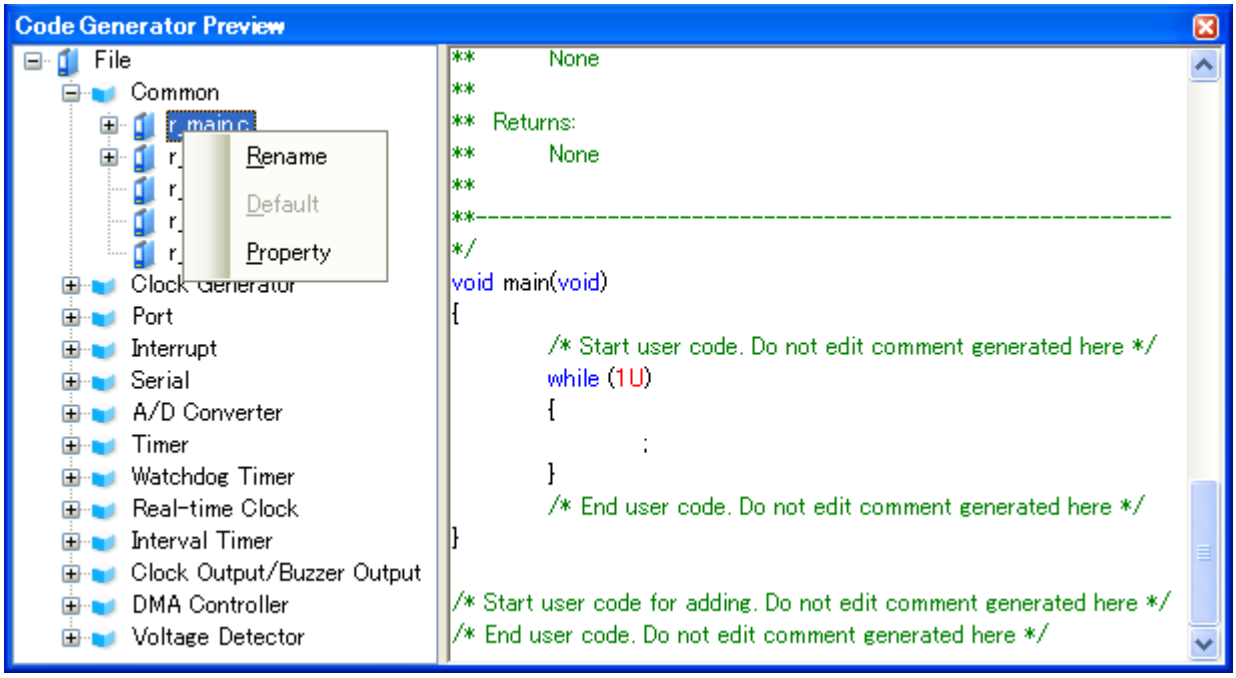
Table 2-4. Setting That Determines Whether or Not to Generate Source Code

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

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

Figure 2-8. Change File Name

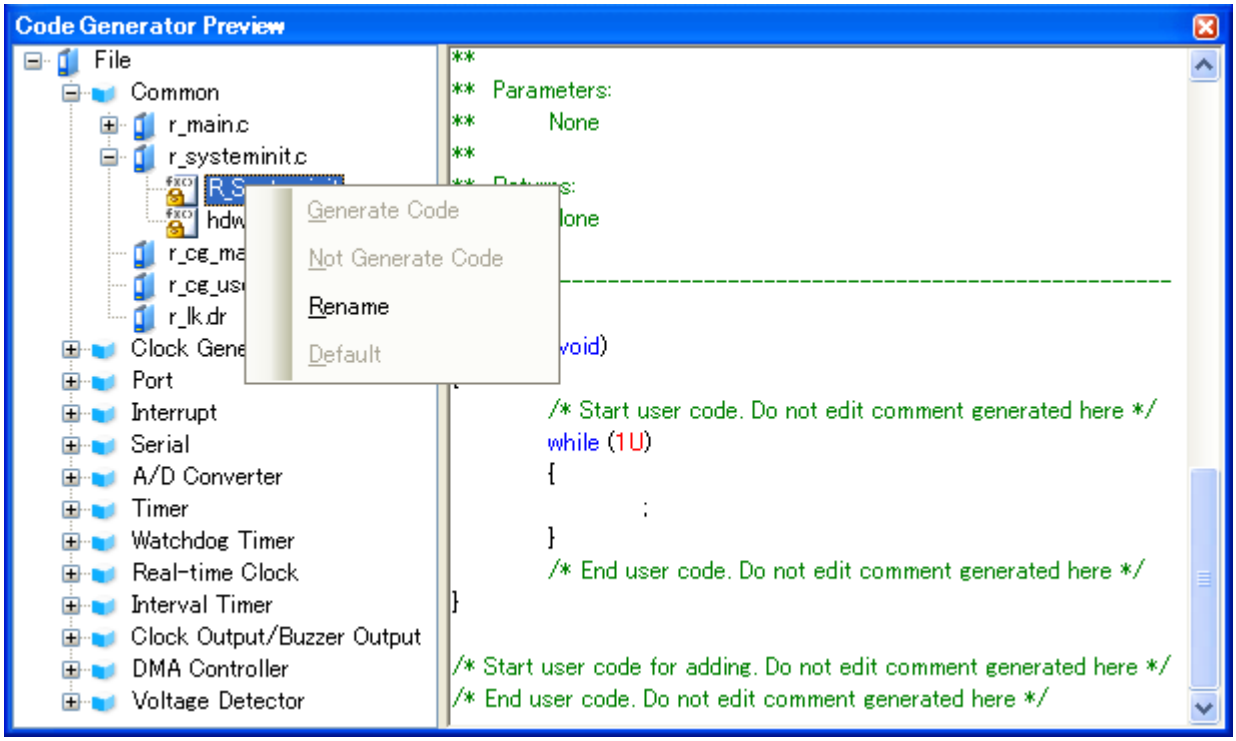


Remark To restore the default file name defined by Code Generator, select [Default] from the context menu.

2.5.3 Change API function name

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

Figure 2-9. Change API Function Name

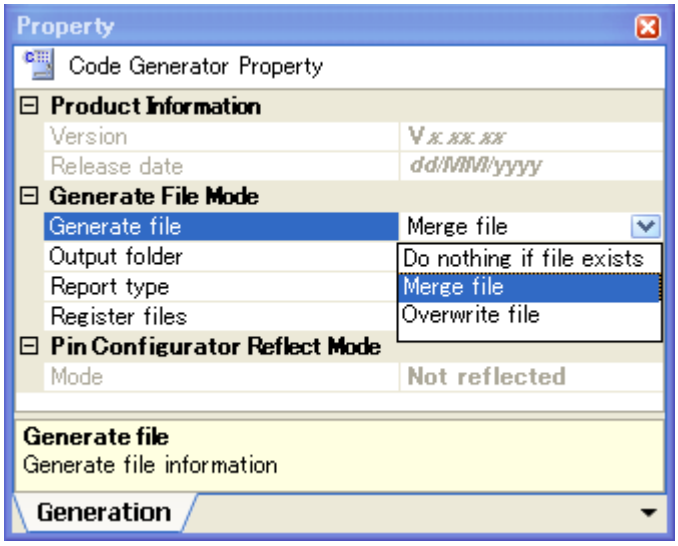


Remark To restore the default name of the API function defined by Code Generator, select [Default] from the context menu.

2.5.4 Change output mode

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

Figure 2-10. Change Output Mode



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

Table 2-5. Output Mode of Source Code

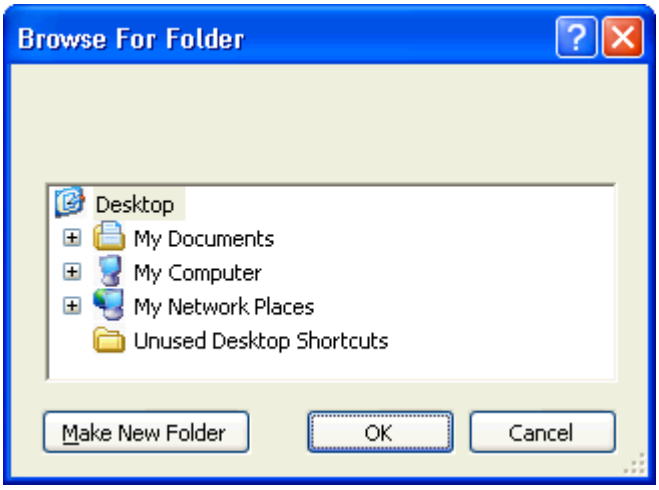
Output Mode	Outline
Overwrite file	If a file with the same name exists, the existing file is overwritten by a new file.
Merge file	If a file with the same name exists, a new file is merged with the existing file. Only the section between "/* Start user code ... . Do not edit comment generated here */" and "/* End user code. Do not edit comment generated here */" will be merged.
Do nothin if file exists	If a file with the same name exists, a new file will not be output.

2.5.5 Change output destination folder

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

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

Figure 2-11. 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 Code Generator panel or Code Generator Preview panel, then selecting [File] menu >> [Save Code Generator Report].

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

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

Table 2-6. Output Report Files

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

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

Figure 2-12. Output Example of Report File "macro"

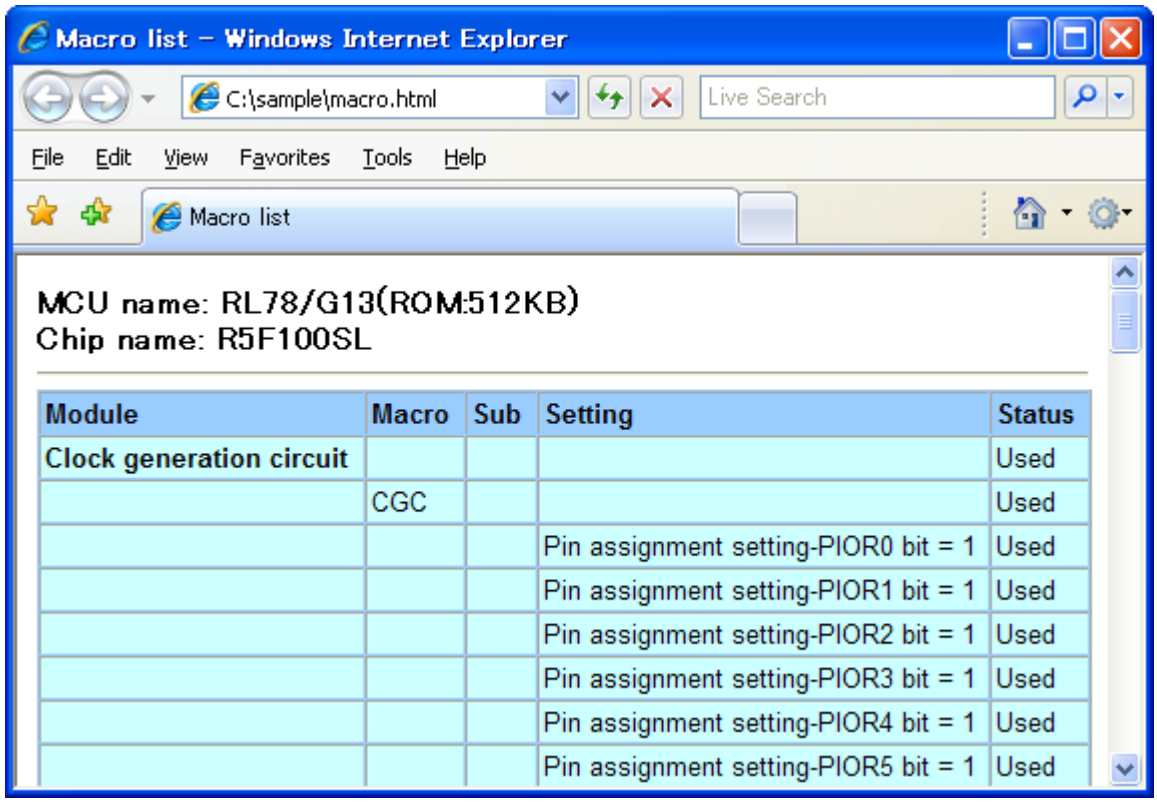
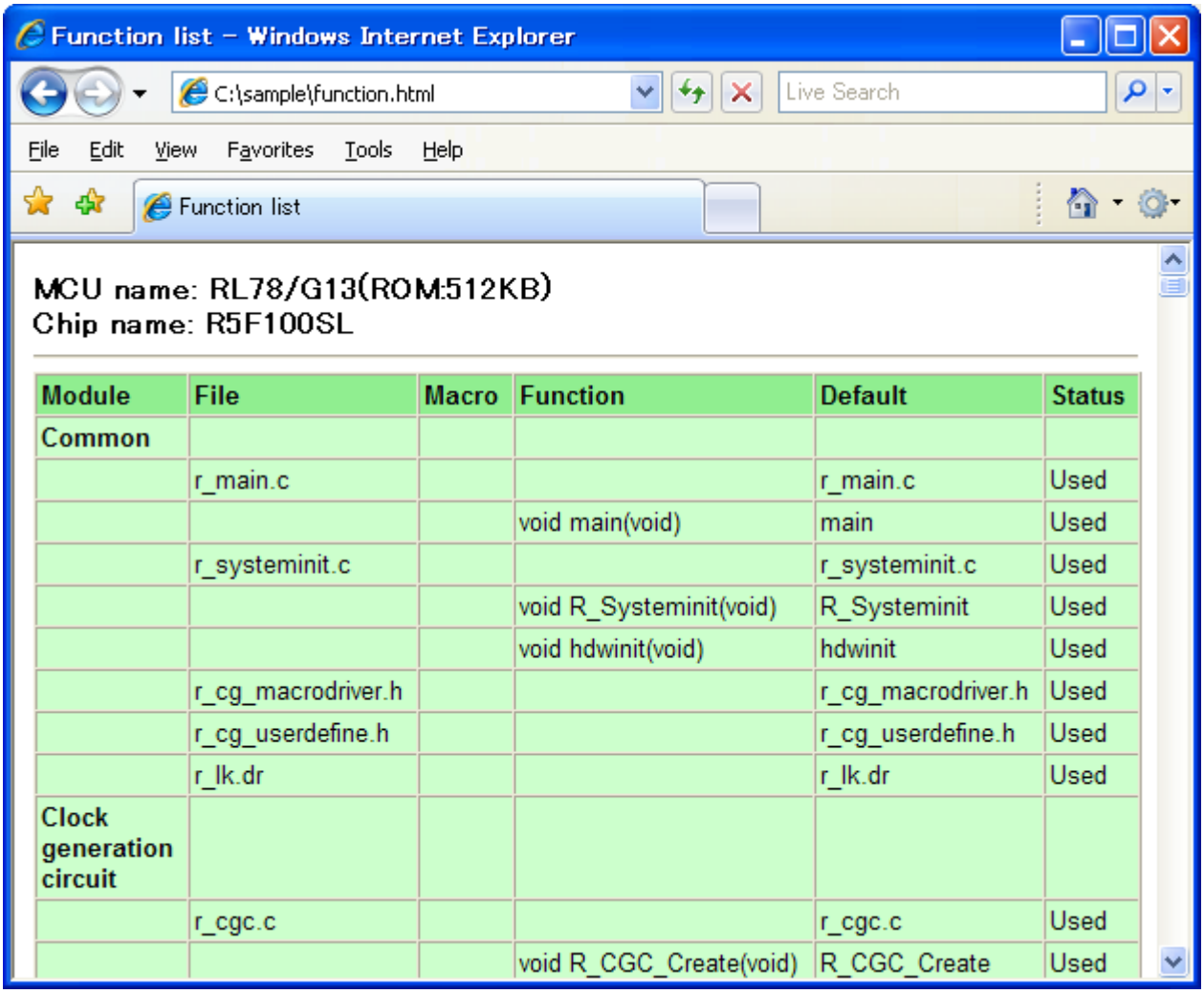


Figure 2-13. Output Example of Report File "function"



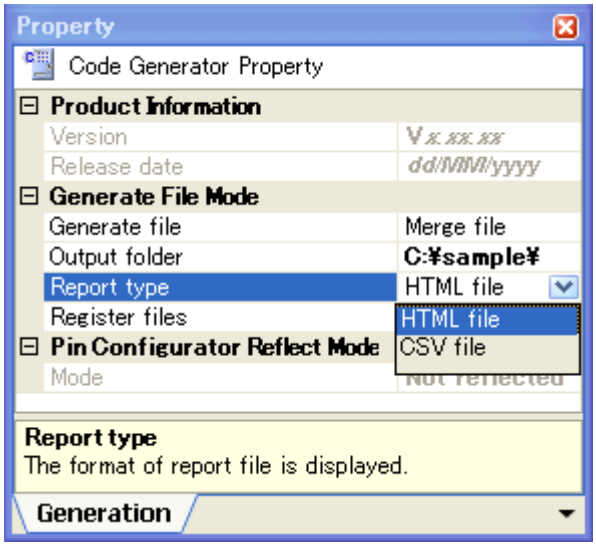
MCU name: RL78/G13(ROM:512KB)
Chip name: R5F100SL

Module	File	Macro	Function	Default	Status
Common					
	r_main.c			r_main.c	Used
			void main(void)	main	Used
	r_systeminit.c			r_systeminit.c	Used
			void R_Systeminit(void)	R_Systeminit	Used
			void hdwinit(void)	hdwinit	Used
	r_cg_macrodriver.h			r_cg_macrodriver.h	Used
	r_cg_userdefine.h			r_cg_userdefine.h	Used
	r_lk.dr			r_lk.dr	Used
Clock generation circuit					
	r_cgc.c			r_cgc.c	Used
			void R_CGC_Create(void)	R_CGC_Create	Used

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 [Generation] tab >> [Report type] in the Property panel.

Figure 2-14. Change Output Format



Remark Output format is selected from the following two types.

Table 2-7. Output Mode of Source Code

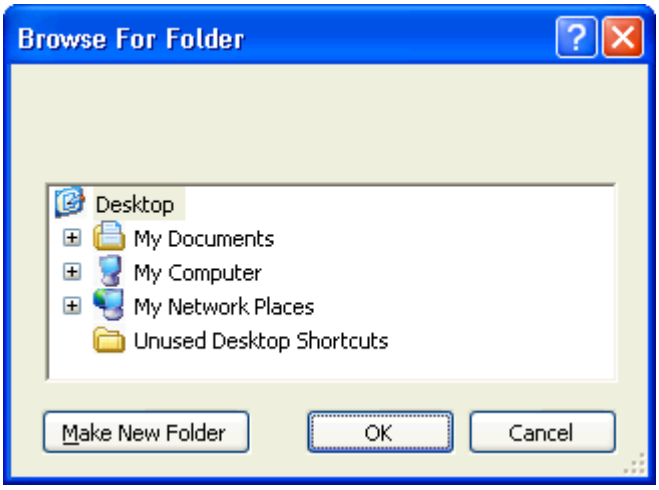
Report Type	Outline
HTML file	Outputs a report file in HTML format.
CSV file	Outputs a report file in CSV format.

2.6.2 Change output destination

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

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

Figure 2-15. Change Output Destination



APPENDIX A WINDOW REFERENCE

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

A.1 Description

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

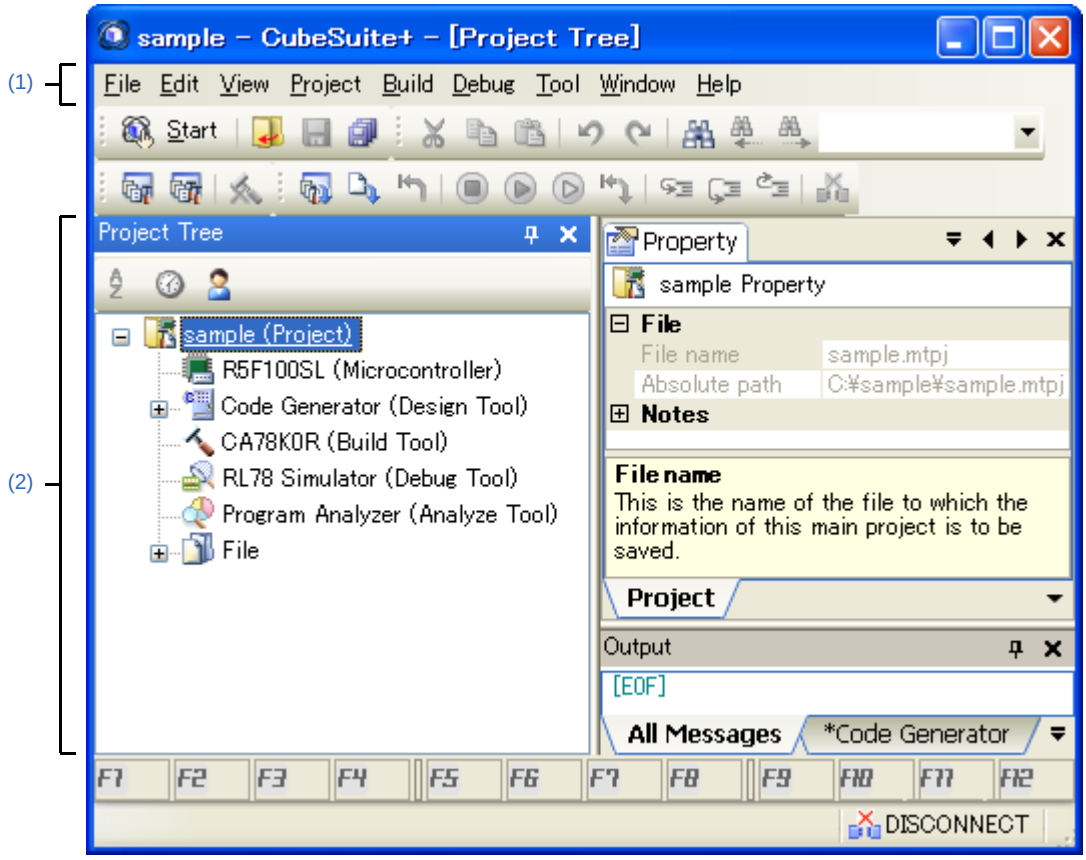
Table A-1. Window/Panel/Dialog Box List

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

Main window

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

Figure A-1. Main Window



The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Menu bar

This area consists of the following menu items.

(a) [File] menu

Save Code Generator Report	Code Generator panel/Code Generator Preview panel-dedicated item Outputs report files (a file containing information configured using Code Generator and a file containing information regarding the source code). - The output format for the report file (either HTML or CSV) is selected by clicking [Generation] tab >> [Report type] in the Property panel. - The destination folder for the report file is specified by clicking [Generation] tab >> [Output folder] in the Property panel.
Save Output-*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...	Code Generator Preview panel/Output panel-dedicated item Opens the Search and Replace dialog box for searching strings with the [Quick Search] tab selected.
Replace...	Output panel-dedicated item Opens the Search and Replace dialog box for replacing strings with the [Whole Replace] tab selected.

(c) [Help] menu

Open Help for Project Tree Panel	Project Tree panel-dedicated item Displays the help of Project Tree panel.
Open Help for Property Panel	Property panel-dedicated item Displays the help of Property panel.

Open Help for Code Generator Panel	Code Generator panel-dedicated item Displays the help of Code Generator panel.
Open Help for Code Generator Preview Panel	Code Generator Preview panel-dedicated item Displays the help of Code Generator Preview panel.
Open Help for Output Panel	Output panel-dedicated item Displays the help of Output panel.

(2) Panel display area

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

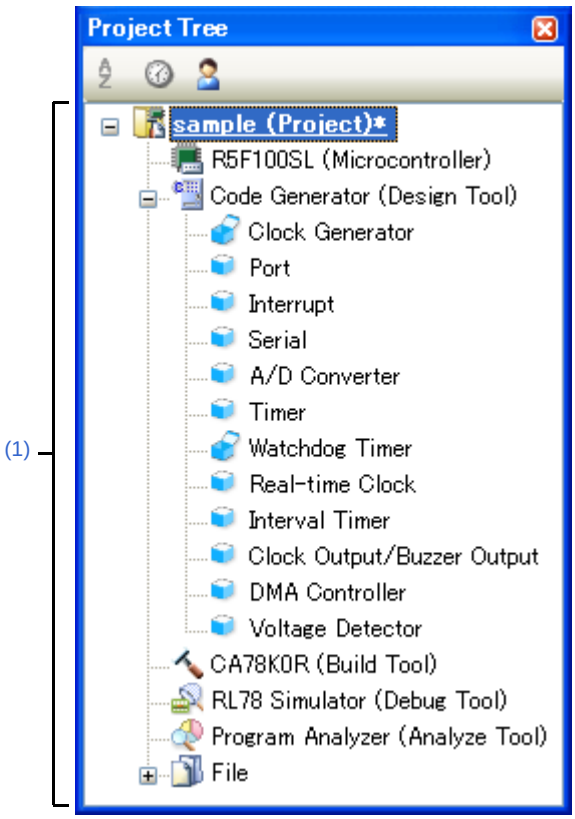
See the following sections for details on this area.

- Project Tree panel
- Property panel
- Code Generator panel
- Code Generator Preview panel
- Output panel

Project Tree panel

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

Figure A-2. Project Tree Panel



The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Project tree area

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

(a) Code Generator (Design Tool)

This node consists of the following peripheral function nodes.
When there is peripheral function target microcontroller is not supporting, peripheral functionbutton is not disokayed.

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

(b) Icons

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

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

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

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

[Context menu]

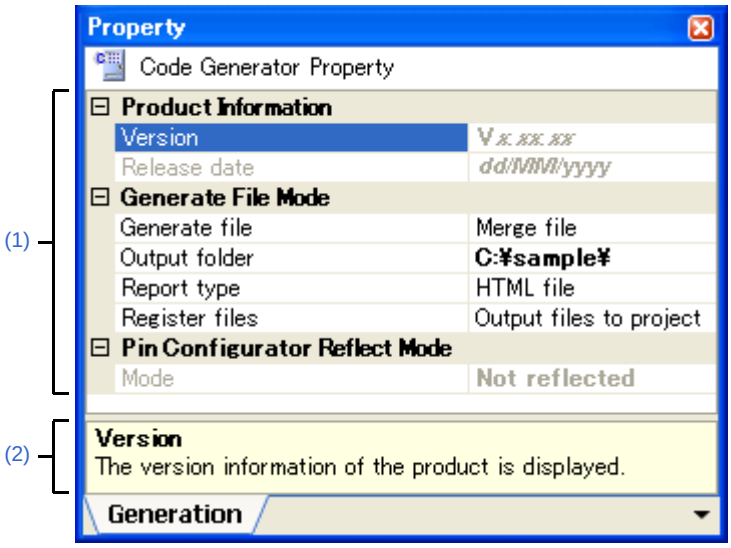
The following context menu items are displayed by right clicking the mouse.

Return to Reset Value	Restores the information for the selected peripheral function node to its default state.
Property	Opens the Property panel containing the information for the selected node ([Code Generator (Design Tool)]).

Property panel

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

Figure A-3. Property Panel (Selected [Code Generator (Design Tool)])



The following items are explained here.

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

[How to open]

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

Remarks 1. If this panel is already open, selecting a different node ([Code Generator (Design Tool)] or peripheral function node such as [Clock Generator], [Port], etc.) in the Project Tree panel changes the content displayed in the Detail information display/change area and Explanation area accordingly.

2. If this panel is already open, pressing a different peripheral function button (such as , , etc.) in the Code Generator panel changes the content displayed in the Detail information display/change area and Explanation area accordingly.

3. If this panel is already open, selecting a different file in the Code Generator Preview panel changes the content displayed in the Detail information display/change area and Explanation area accordingly.

[Description of each area]

(1) Detail information display/change area

This area allows you to view the information on and change the setting for the node ([Code Generator (Design Tool)] or peripheral function node such as [Clock Generator], [Port], etc.) selected in the Project Tree panel, the peripheral function button (such as  ,  , etc.) pressed in the Code Generator panel, or the file selected in the Code Generator Preview panel.

The content displayed in this area differs depending on the node selected in the Project Tree panel, the peripheral function button pressed in the Code Generator panel or the file selected in the Code Generator Preview panel.

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

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

Remarks 1. See the sections "[Generation] tab", "[Macro Setting] tab" and "[File Setting] tab" for details on the content displayed in this area.

2. To switch between ⊞ and ⊟ , click this mark or double-click the category name.

(2) Explanation area

This area displays a "brief description" of the category or item selected in the Detail information display/change area.

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

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

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

Open Help for Property Panel	Displays the help of this panel.

[Context menu]

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

(1) While the item is being edited

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

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

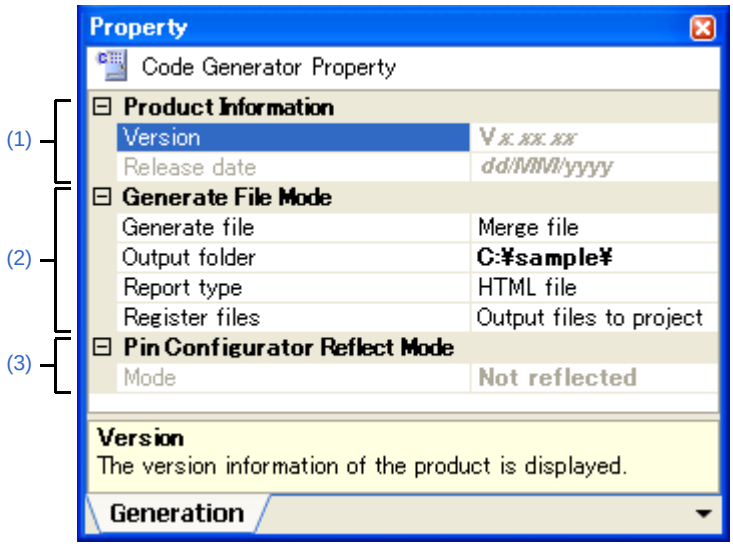
(2) While the item is not being edited

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

[Generation] tab

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

Figure A-4. [Generation] Tab



The following items are explained here.

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

[How to open]

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

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

[Description of each area]

(1) [Product Information] category

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

Version	Displays the version of Code Generator.
Release date	Displays the release date of Code Generator.

(2) [Generate File Mode] category

This area allows you to view and change the setting for the file generation mode (Generate file, Output folder, Report type and Register files) of Code Generator.

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

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

(3) [Pin Configurator Reflect Mode] category

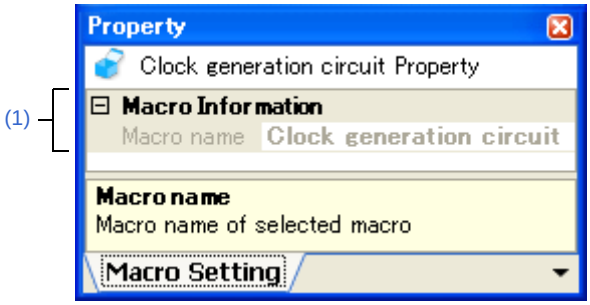
Not supported in this version.

This category will be grayed out.

[Macro Setting] tab

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

Figure A-5. [Macro Setting] Tab



The following items are explained here.

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

[How to open]

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

Remarks 1. If this panel is already open, selecting a different peripheral function node "[Clock Generator], [Port], etc." in the Project Tree panel changes the content displayed accordingly.

2. If this panel is already open, pressing a different type of peripheral function button " , , etc." in the Code Generator panel changes the content displayed accordingly.

[Description of each area]

(1) [Macro Information] category

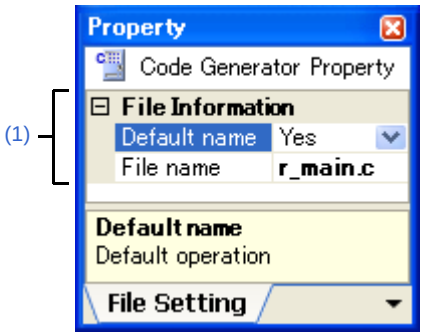
This area allows you to view the information (Macro name) on and change the setting for the peripheral function node "[Clock Generator], [Port], etc." selected in the Project Tree panel, or the peripheral function button pressed in the Code Generator panel.

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

[File Setting] tab

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

Figure A-6. [File Setting] Tab



The following items are explained here.

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

[How to open]

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

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

[Description of each area]

(1) [File Information] category

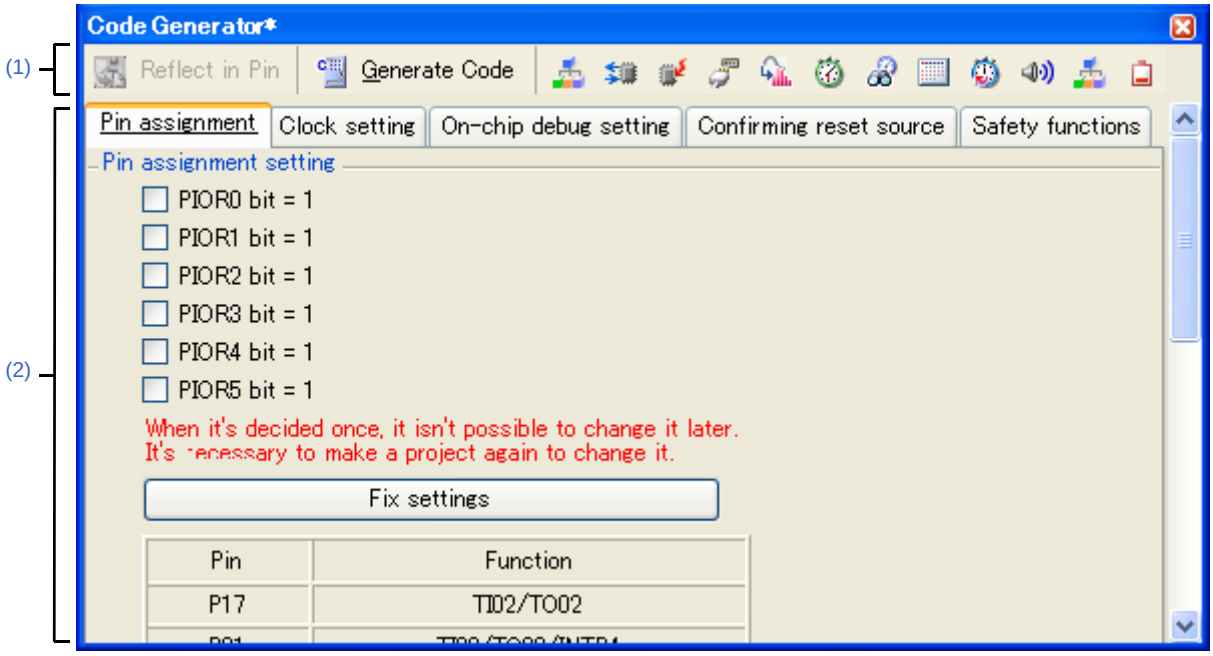
This area allows you to view the information (Default, File name) on and change the setting for the file selected in the Code Generator Preview panel.

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

Code Generator panel

This panel allows you to configure the information necessary to control the peripheral functions provided by the micro-controller.

Figure A-7. Code Generator Panel: [Clock Generator]



The following items are explained here.

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

[How to open]

- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> Peripheral function node "[Clock Generator], [Port], etc.".

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

[Description of each area]

(1) Toolbar

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

When there is peripheral function target microcontroller is not supporting, peripheral functionbutton is not disokayed.

Reflect in Pin	Not supported in this version. This button will be grayed out (disabled).

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

	Changes the content displayed in the Information setting area to the "[Voltage Detector] for configuring the information necessary to control the functions of voltage detector provided by the microcontroller".

(2) Information setting area

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

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

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

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

Remarks 1. The output format for the report file (either HTML or CSV) is selected by clicking [Generation] tab >> [Report type] in the Property panel.

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

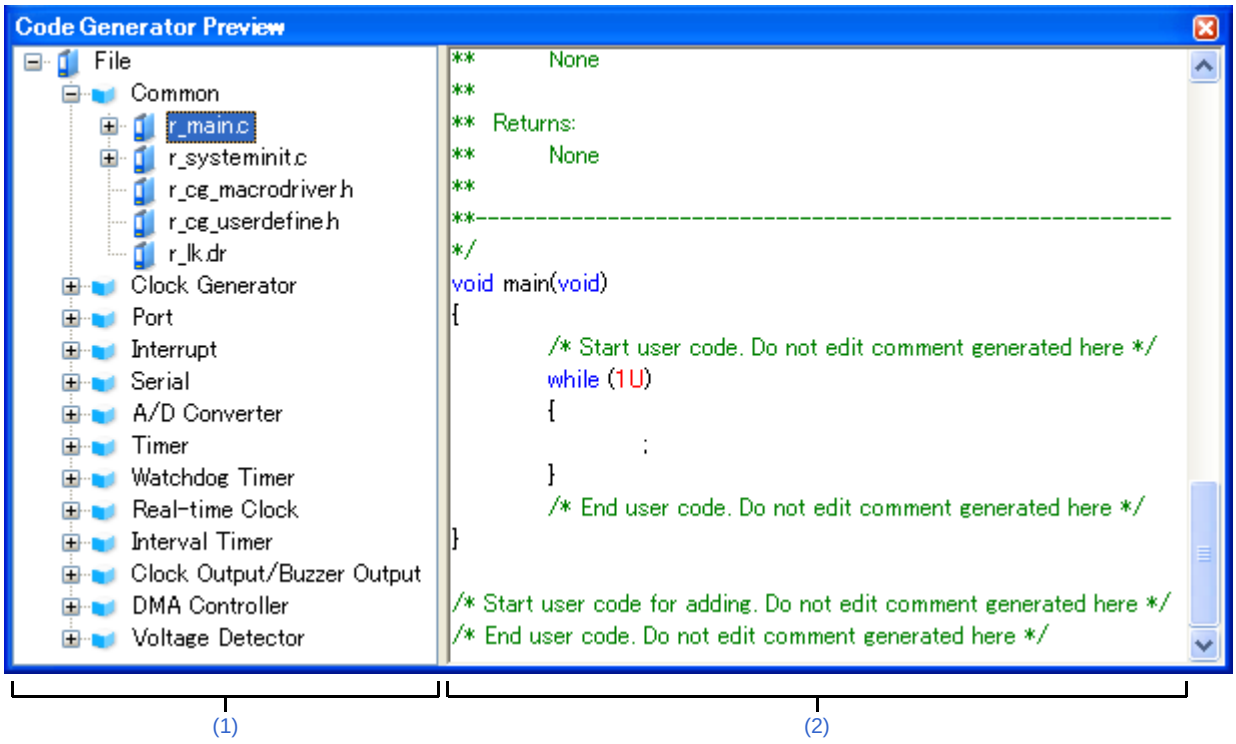
[[Help] menu (Code Generator panel-dedicated items)]

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

Code Generator Preview panel

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

Figure A-8. Code Generator Preview Panel



The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Preview tree

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

Remarks 1. You can change the source code to be displayed by selecting the source file name or API function name in this tree.

2. To select whether or not to generate the source code, use the context menu (Generate code/Not generate code) which is displayed by right-clicking the mouse while the mouse cursor is on the desired icon in the tree.

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

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

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

(2) Source code display area

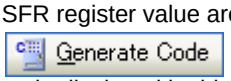
This area allows you to confirm the source code (device driver program) that reflects the information configured in the Code Generator panel.

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

Table A-3. Color of Source Code

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

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

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

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

3. You can change the source code to be displayed by selecting the source file name or API function name in the preview tree.

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

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

Remarks 1. The output format for the report file (either HTML or CSV) is selected by clicking [Generation] tab >> [Report type] in the Property panel.

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

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

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

[Context menu]

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

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

Output panel

This panel displays operation logs for various components (design tool, build tool, etc.) provided by CubeSuite+.

Figure A-9. Output Panel

The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Message area

This area displays operation logs for various components (design tool, build tool, etc.) provided by CubeSuite+.
The following table displays the meaning of the color of the message text displayed in this area.

Table A-4. Color of Message Text/Background

Message Text/Background	Description
Block/White	Information message Displayed with information notices.
Blue/Standard color	Warning message Displayed with warnings about operations.
Red/LightGray	Fatal error message Displayed when there is a fatal error, or when execution is not possible due to a operational mistake.

Remark See the sections "[All Output Messages] tab" and "[Code Generator] tab" for details on the content displayed in this area.

(2) Tab selection area

Select the source of message.

Remark When the new message is output, "*" mark is displayed to the left of the tab name.

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

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

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

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

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

Open Help for Output Panel	Displays the help of this panel.

[Context menu]

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

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

[All Output Messages] tab

This tab displays operation logs for all components (design tool, build tool, etc.) provided by CubeSuite+.

Figure A-10. [All Output Messages] Tab

The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Message area

This area displays operation logs for all components (design tool, build tool, etc.) provided by CubeSuite+.
The following table displays the meaning of the color of the message text displayed in this area.

Table A-5. Color of Message Text/Background

Message Text/Background	Description
Black/White	Information message Displayed with information notices.
Blue/Standard Color	Warning message Displayed with warnings about operations.
Red/LightGray	Fatal error message Displayed when there is a fatal error, or when execution is not possible due to a operational mistake.

[Code Generator] tab

This tab displays only operation logs for Code Generator out of those for various components (design tool, build tool, etc.) provided by CubeSuite+.

Figure A-11. [Code Generator] Tab

The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Message area

This area displays only operation logs for Code Generator out of those for various components (design tool, build tool, etc.) provided by CubeSuite+.

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

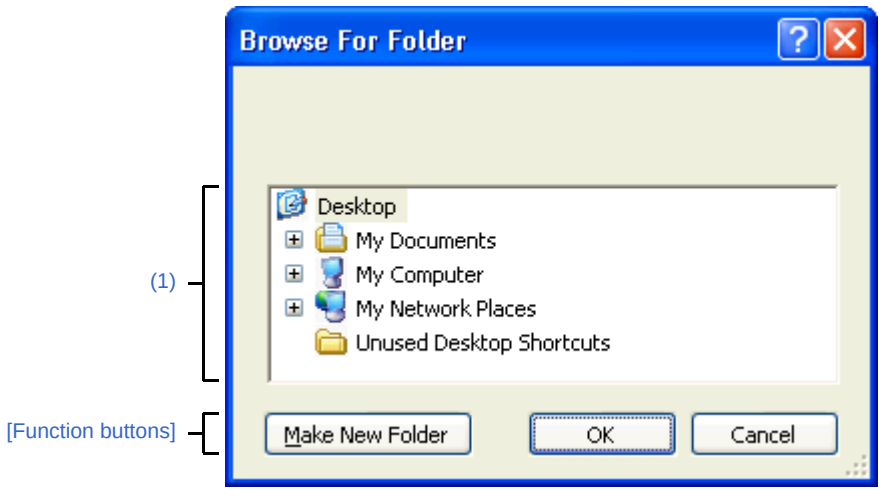
Table A-6. Color of Message Text/Background

Message Text/Background	Outline
Black/White	Information message Displayed with information notices.
Blue/Standard Color	Warning message Displayed with warnings about operations.
Red/LightGray	Fatal error message Displayed when there is a fatal error, or when execution is not possible due to a operational mistake.

Browse For Folder dialog box

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

Figure A-12. Browse For Folder Dialog Box



The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Folder location

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

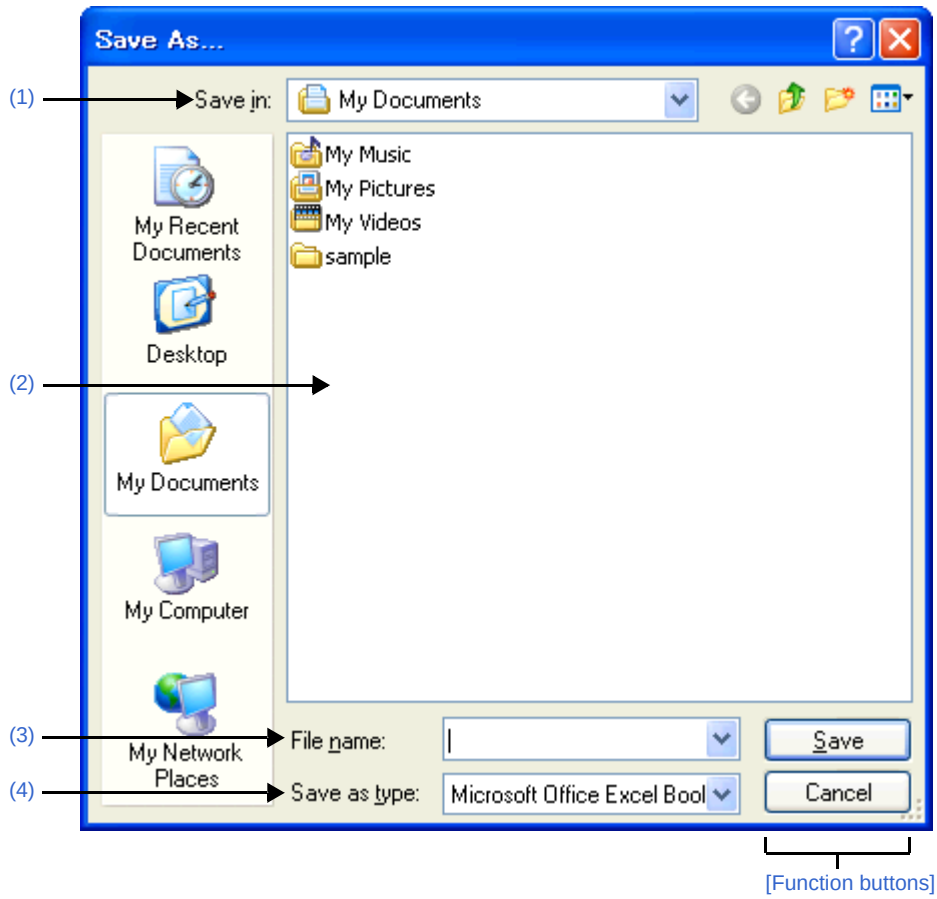
[Function buttons]

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

Save As dialog box

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

Figure A-13. Save As Dialog Box



The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) [Save in]

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

(2) List of files

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

(3) [File name]

Specify the name of the file to be output.

(4) [Save as type]

Select the type of the file to be output.

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

[Function buttons]

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

APPENDIX B OUTPUT FILES

This appendix describes the files output by the Code Generator.

B.1 Overview

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

Table B-1. File List

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

B.2 Output File

Below are the files (peripheral function) output by the Code Generator.

Table B-2. File List (Peripheral Function)

Peripheral Function	Source File Name	Names of API Functions Included
Clock Generator	r_cgc.c	R_CGC_Create R_CGC_Set_ClockMode R_CGC_Set_CRCOn
	r_cgc_user.c	R_CGC_Create_UserInit R_CGC_Get_ResetSource
	r_cg_cgc.h	-
Port	r_port.c	R_PORT_Create
	r_port_user.c	R_PORT_Create_UserInit
	r_cg_port.h	-
Interrupt	r_intc.c	R_INTC_Create R_INTCn_Start R_INTCn_Stop R_KEY_Create R_KEY_Start R_KEY_Stop
	r_intc_user.c	R_INTC_Create_UserInit R_INTCn_Interrupt R_KEY_Create_UserInit R_KEY_Interrupt

Peripheral Function	Source File Name	Names of API Functions Included
Interrupt	r_cg_intc.h	-
Serial	r_serial.c	R_SAUm_Create R_SAUm_Set_PowerOff R_SAU0_Set_SnoozeOn R_SAU0_Set_SnoozeOff R_UARTn_Create R_UARTn_Start R_UARTn_Stop R_UARTn_Send R_UARTn_Receive R_CSImn_Create R_CSImn_Start R_CSImn_Stop R_CSImn_Send R_CSImn_Receive R_CSImn_Send_Receive R_IICmn_Create R_IICmn_StartCondition R_IICmn_StopCondition R_IICmn_Stop R_IICmn_Master_Send R_IICmn_Master_Receive R_IICAn_Create R_IICAn_StopCondition R_IICAn_Stop R_IICAn_Set_PowerOff R_IICAn_Master_Send R_IICAn_Master_Receive R_IICAn_Slave_Send R_IICAn_Slave_Receive
	r_serial_user.c	R_SAUm_Create_UserInit R_UARTn_Interrupt_Send R_UARTn_Interrupt_Receive R_UARTn_Interrupt_Error R_UARTn_Callback_SendEnd R_UARTn_Callback_ReceiveEnd R_UARTn_Callback_Error R_UARTn_Callback_SoftwareOverRun R_CSImn_Interrupt R_CSImn_Callback_SendEnd R_CSImn_Callback_ReceiveEnd R_CSImn_Callback_Error R_IICmn_Interrupt R_IICmn_Callback_Master_SendEnd R_IICmn_Callback_Master_ReceiveEnd R_IICmn_Callback_Master_Error R_IICAn_Create_UserInit

Peripheral Function	Source File Name	Names of API Functions Included
Serial	r_serial_user.c	R_IICAn_Interrupt R_IICAn_Callback_Master_SendEnd R_IICAn_Callback_Master_ReceiveEnd R_IICAn_Callback_Master_Error R_IICAn_Callback_Slave_SendEnd R_IICAn_Callback_Slave_ReceiveEnd R_IICAn_Callback_Slave_Error R_IICAn_Callback_GetStopCondition
	r_cg_serial.h	-
A/D Converter	r_adc.c	R_ADC_Create R_ADC_Set_ComparatorOn R_ADC_Set_ComparatorOff R_ADC_Start R_ADC_Stop R_ADC_Set_PowerOff R_ADC_Set_ADChannel R_ADC_Set_SnoozeOn R_ADC_Set_SnoozeOff R_ADC_Set_TestChannel R_ADC_Get_Result R_ADC_Get_Result_8bit
	r_adc_user.c	R_ADC_Create_UserInit R_ADC_Interrupt
	r_cg_adc.h	-
D/A Converter	r_dac.c	R_DAC_Create R_DACn_Start R_DACn_Stop R_DAC_Set_PowerOff R_DACn_Set_ConversionValue
	r_dac_user.c	R_DAC_Create_UserInit
	r_cg_dac.h	-
Timer	r_timer.c	R_TAUm_Create R_TAUm_Channeln_Start R_TAUm_Channeln_Higher8bits_Start R_TAUm_Channeln_Lower8bits_Start R_TAUm_Channeln_Stop R_TAUm_Channeln_Higher8bits_Stop R_TAUm_Channeln_Lower8bits_Stop R_TAUm_Set_PowerOff R_TAUm_Channeln_Get_PulseWidth R_TAUm_Channeln_SoftwareTriggerOn R_TMR_RDn_Create R_TMR_RDn_Start R_TMR_RDn_Stop R_TMR_RDn_Set_PowerOff

Peripheral Function	Source File Name	Names of API Functions Included
Timer	r_timer.c	R_TMR_RDn_Get_PulseWidth R_TMR_RGn_Create R_TMR_RGn_Start R_TMR_RGn_Stop R_TMR_RGn_Set_PowerOff R_TMR_RGn_Get_PulseWidth R_TMR_RJ0_Create R_TMR_RJ0_Start R_TMR_RJ0_Stop R_TMR_RJ0_Set_PowerOff R_TMR_RJ0_Get_PulseWidth
	r_timer_user.c	R_TAUm_Create_UserInit R_TAUm_Channeln_Interrupt R_TAUm_Channeln_Higher8bits_Interrupt R_TMR_RDn_Create_UserInit R_TMR_RDn_Interrupt R_TMR_RGn_Create_UserInit R_TMR_RGn_Interrupt R_TMR_RJ0_Create_UserInit R_TMR_RJ0_Interrupt
	r_cg_timer.h	-
Watchdog Timer	r_wdt.c	R_WDT_Create R_WDT_Restart
	r_wdt_user.c	R_WDT_Create_UserInit R_WDT_Interrupt
	r_cg_wdt.h	-
Real-time Clock	r_rtc.c	R_RTC_Create R_RTC_Start R_RTC_Stop R_RTC_Set_PowerOff R_RTC_Set_HourSystem R_RTC_Set_CounterValue R_RTC_Get_CounterValue R_RTC_Set_ConstPeriodInterruptOn R_RTC_Set_ConstPeriodInterruptOff R_RTC_Set_AlarmOn R_RTC_Set_AlarmOff R_RTC_Set_AlarmValue R_RTC_Get_AlarmValue R_RTC_Set_RTC1HZOn R_RTC_Set_RTC1HZOff
	r_rtc_user.c	R_RTC_Create_UserInit R_RTC_Interrupt R_RTC_Callback_ConstPeriod R_RTC_Callback_Alarm

Peripheral Function	Source File Name	Names of API Functions Included
Rel-time Clock	r_cg_rtc.h	-
Interval Timer	r_it.c	R_IT_Create R_IT_Start R_IT_Stop R_IT_Set_PowerOff
	r_it_user.c	R_IT_Create_UserInit R_IT_Interrupt
	r_cg_it.h	-
Comparator	r_comp.c	R_COMP_Create R_COMPn_Start R_COMPn_Stop
	r_comp_user.c	R_COMP_Create_UserInit R_COMPn_Interrupt
	r_cg_comp.h	-
Clock Output/Buzzer Output	r_pclbuz.c	R_PCLBUZn_Create R_PCLBUZn_Start R_PCLBUZn_Stop
	r_pclbuz_user.c	R_PCLBUZn_Create_UserInit
	r_cg_pclbuz.h	-
Data Transfer Controller	r_dtc.c	R_DTC_Create R_DTCn_Start R_DTCn_Stop R_DTC_Set_PowerOff
	r_dtc_user.c	R_DTC_Create_UserInit
	r_cg_dtc.h	-
Event Link Controller	r_elc.c	R_ELC_Create R_ELC_Stop
	r_elc_user.c	R_ELC_Create_UserInit
	r_cg_elc.h	-
DMA Controller	r_dmac.c	R_DMACn_Create R_DMACn_Start R_DMACn_Stop R_DMACn_SoftwareTriggerOn
	r_dmac_user.c	R_DMACn_Create_UserInit R_DMACn_Interrupt
	r_cg_dmac.h	-
Voltage Detector	r_lvd.c	R_LVD_Create R_LVD_InterruptMode_Start
	r_lvd_user.c	R_LVD_Create_UserInit R_LVD_Interrupt
	r_cg_lvd.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 local variables start with a "p" and are in all lower case.
- Names of pointers to global variables start with a "gp" and use Camel Case.
- Names of elements in enum statements are in ALL CAPS.

C.2 Output Function

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

Table C-1. API Function List

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

Peripheral Function	API Function Name	Function
Interrupt	R_KEY_Start	Enables the acceptance of the key interrupts INTKR.
	R_KEY_Stop	Disables the acceptance of the key interrupts INTKR.
Serial	R_SAUm_Create	Performs initialization necessary to control the serial array unit and serial interface functions.
	R_SAUm_Create_UserInit	Performs user-defined initialization related to the serial array unit and serial interface functions.
	R_SAUm_Set_PowerOff	Halts the clock supplied to the serial array unit.
	R_SAU0_Set_SnoozeOn	Enables the switch from STOP mode to SNOOZE mode.
	R_SAU0_Set_SnoozeOff	Disables the switch from STOP mode to SNOOZE mode.
	R_UARTn_Create	Performs initialization of the serial interface (UART) channel.
	R_UARTn_Interrupt_Send	Performs processing in response to the UART transmission end interrupt INTST*n*.
	R_UARTn_Interrupt_Receive	Performs processing in response to the UART reception end interrupt INTSR*n*.
	R_UARTn_Interrupt_Error	Performs processing in response to the UART reception error interrupt INTSRE*n*.
	R_UARTn_Start	Sets UART communication to standby mode.
	R_UARTn_Stop	Ends UART communication.
	R_UARTn_Send	Starts UART data transmission.
	R_UARTn_Receive	Starts UART data reception.
	R_UARTn_Callback_SendEnd	Performs processing in response to the UART transmission end interrupt INTST*n*.
	R_UARTn_Callback_ReceiveEnd	Performs processing in response to the UART reception cend interrupt INTSR*n*.
	R_UARTn_Callback_Error	Performs processing in response to the reception error interrupt INTSRE*n*.
	R_UARTn_Callback_SoftwareOverRun	Performs processing in response to detection of overrun error.
	R_CSImn_Create	Performs initialization of the serial interface (CSI) channel.
	R_CSImn_Interrupt	Performs processing in response to the CSI communication end interrupt INTCSI*mn*.
	R_CSImn_Start	Sets CSI communication to standby mode.
	R_CSImn_Stop	Ends CSI communication.
	R_CSImn_Send	Starts CSI data transmission.
	R_CSImn_Receive	Starts CSI data reception.
	R_CSImn_Send_Receive	Starts CSI data transmission/reception.
	R_CSImn_Callback_SendEnd	Performs processing in response to the CSI transmission end interrupt INTCSI*mn*.

Peripheral Function	API Function Name	Function
Serial	R_CSImn_Callback_ReceiveEnd	Performs processing in response to the CSI reception end interrupt INTCSI*mn*.
	R_CSImn_Callback_Error	Performs processing in response to the CSI reception error interrupt INTSRE*n*.
	R_IICmn_Create	Performs initialization of the serial interface (simple IIC) channel.
	R_IICmn_Interrupt	Performs processing in response to the simple IIC communication end interrupt INTIIC*mn*.
	R_IICmn_StartCondition	Generates start conditions.
	R_IICmn_StopCondition	Generates stop conditions.
	R_IICmn_Stop	Ends simple IIC communication.
	R_IICmn_Master_Send	Starts simple IIC master transmission.
	R_IICmn_Master_Receive	Starts simple IIC master reception.
	R_IICmn_Callback_Master_SendEnd	Performs processing in response to the simple IIC*mn* master transmission end interrupt INTIIC*mn*.
	R_IICmn_Callback_Master_ReceiveEnd	Performs processing in response to the simple IIC*mn* master reception end interrupt INTIIC*mn*.
	R_IICmn_Callback_Master_Error	Performs processing in response to detection of parity error (ACK error).
	R_IICAn_Create	Performs initialization of the serial interface (IICA).
	R_IICAn_Create_UserInit	Performs user-defined initialization of the serial interface (IICA).
	R_IICAn_Interrupt	Performs processing in response to the IICA communication end interrupt INTIICA*n*.
	R_IICAn_StopCondition	Generates stop conditions.
	R_IICAn_Stop	Ends IICA communication.
	R_IICAn_Set_PowerOff	Halts the clock supplied to the serial interface (IICA).
	R_IICAn_Master_Send	Starts IICA master transmission.
	R_IICAn_Master_Receive	Starts IICA master reception.
	R_IICAn_Callback_Master_SendEnd	Performs processing in response to the IICA master transmission end interrupt INTIICA*n*.
	R_IICAn_Callback_Master_ReceiveEnd	Performs processing in response to the IICA master reception end interrupt INTIICA*n*.
	R_IICAn_Callback_Master_Error	Performs processing in response to detection of IICA master communication error.
	R_IICAn_Slave_Send	Starts IICA slave transmission.
	R_IICAn_Slave_Receive	Starts IICA slave reception.
	R_IICAn_Callback_Slave_SendEnd	Performs processing in response to the IICA slave transmission end interrupt INTIICA*n*.
	R_IICAn_Callback_Slave_ReceiveEnd	Performs processing in response to the IICA slave reception end interrupt INTIICA*n*.

Peripheral Function	API Function Name	Function
Serial	R_IICAn_Callback_Slave_Error	Performs processing in response to detection of IICA slave communication error.
	R_IICAn_Callback_GetStopCondition	Performs processing in response to detection of stop condition.
A/D Converter	R_ADC_Create	Performs initialization necessary to control A/D converter functions.
	R_ADC_Create_UserInit	Performs user-defined initialization relating to the A/D converter.
	R_ADC_Interrupt	Performs processing in response to the A/D conversion end interrupt INTAD.
	R_ADC_Set_ComparatorOn	Enables operation of voltage converter.
	R_ADC_Set_ComparatorOff	Disables operation of voltage converter.
	R_ADC_Start	Starts A/D conversion.
	R_ADC_Stop	Ends A/D conversion.
	R_ADC_Set_PowerOff	Halts the clock supplied to the A/D converter.
	R_ADC_Set_ADChannel	Configures the analog voltage input pin for A/D conversion.
	R_ADC_Set_SnoozeOn	Enables the switch from STOP mode to SNOOZE mode.
	R_ADC_Set_SnoozeOff	Disables the switch from STOP mode to SNOOZE mode.
	R_ADC_Set_TestChannel	Sets the operaiton mode of A/D converter.
	R_ADC_Get_Result	Reads the results of A/D conversion (10-bit).
	R_ADC_Get_Result_8bit	Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).
D/A Converter	R_DAC_Create	Performs initialization necessary to control the D/A converter functions.
	R_DAC_Create_UserInit	Performs user-defined initialization relating to the D/A converter.
	R_DACn_Start	Starts D/A conversion.
	R_DACn_Stop	Ends D/A conversion.
	R_DAC_Set_PowerOff	Halts the clock supplied to the D/A converter.
	R_DACn_Set_ConversionValue	Sets the analog voltage output to the ANO*n* pin.
Timer	R_TAUm_Create	Performs initialization necessary to control timer array unit functions.
	R_TAUm_Create_UserInit	Performs user-defined initialization relating to the timer array unit.
	R_TAUm_Channeln_Interrupt	Performs processing in response to the timer interrupt INTTM*mn*.
	R_TAUm_Channeln_Higher8bits_Interrupt	Performs processing in response to the timer interrupt INTTM*mn*H.
	R_TAUm_Channeln_Start	Starts the count for channel *n*.

Peripheral Function	API Function Name	Function
Timer	R_TAUm_Channeln_Higher8bits_Start	Starts the count (higher 8-bit) for channel 1 or channel 3.
	R_TAUm_Channeln_Lower8bits_Start	Starts the count (lower 8-bit) for channel 1 or channel 3.
	R_TAUm_Channeln_Stop	Ends the count for channel *n*.
	R_TAUm_Channeln_Higher8bits_Stop	Ends the count (higher 8-bit) for channel 1 or channel 3.
	R_TAUm_Channeln_Lower8bits_Stop	Ends the count (lower 8-bit) for channel 1 or channel 3.
	R_TAUm_Set_PowerOff	Halts the clock supplied to the timer array unit.
	R_TAUm_Channeln_Get_PulseWidth	Captures the high/low-level width measured between pulses of the signal (pulses) input to the TI*mn* pin.
	R_TAUm_Channeln_SoftwareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.
	R_TMR_RDn_Create	Performs initialization necessary to control the timer RD*n* functions.
	R_TMR_RDn_Create_UserInit	Performs user-defined initialization relating to the timer RD*n*.
	R_TMR_RDn_Interrupt	Performs processing in response to the timer interrupt.
	R_TMR_RDn_Start	Starts the count for timer RD*n*.
	R_TMR_RDn_Stop	Ends the count for timer RD*n*.
	R_TMR_RDn_Set_PowerOff	Halts the clock supplied to the timer RD*n*.
	R_TMR_RDn_Get_PulseWidth	Reads the pulse width of the timer RD*n*.
	R_TMR_RGn_Create	Performs initialization necessary to control the timer RG*n* functions.
	R_TMR_RGn_Create_UserInit	Performs user-defined initialization relating to the timer RG*n*.
	R_TMR_RGn_Interrupt	Performs processing in response to the timer interrupt.
	R_TMR_RGn_Start	Starts the count for timer RG*n*.
	R_TMR_RGn_Stop	Ends the count for timer RG*n*.
	R_TMR_RGn_Set_PowerOff	Halts the clock supplied to the timer RG*n*.
	R_TMR_RGn_Get_PulseWidth	Reads the pulse width of the timer RG*n*.
	R_TMR_RJ0_Create	Performs initialization necessary to control the timer RJ0 functions.
	R_TMR_RJ0_Create_UserInit	Performs user-defined initialization relating to the timer RJ0.
	R_TMR_RJ0_Interrupt	Performs processing in response to the timer interrupt.
	R_TMR_RJ0_Start	Starts the count for timer RJ0.
	R_TMR_RJ0_Stop	Ends the count for timer RJ0.
	R_TMR_RJ0_Set_PowerOff	Halts the clock supplied to the timer RJ0.
	R_TMR_RJ0_Get_PulseWidth	Reads the pulse width of the timer RJ0.
Watchdog Timer	R_WDT_Create	Performs initialization necessary to control watchdog timer functions.

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

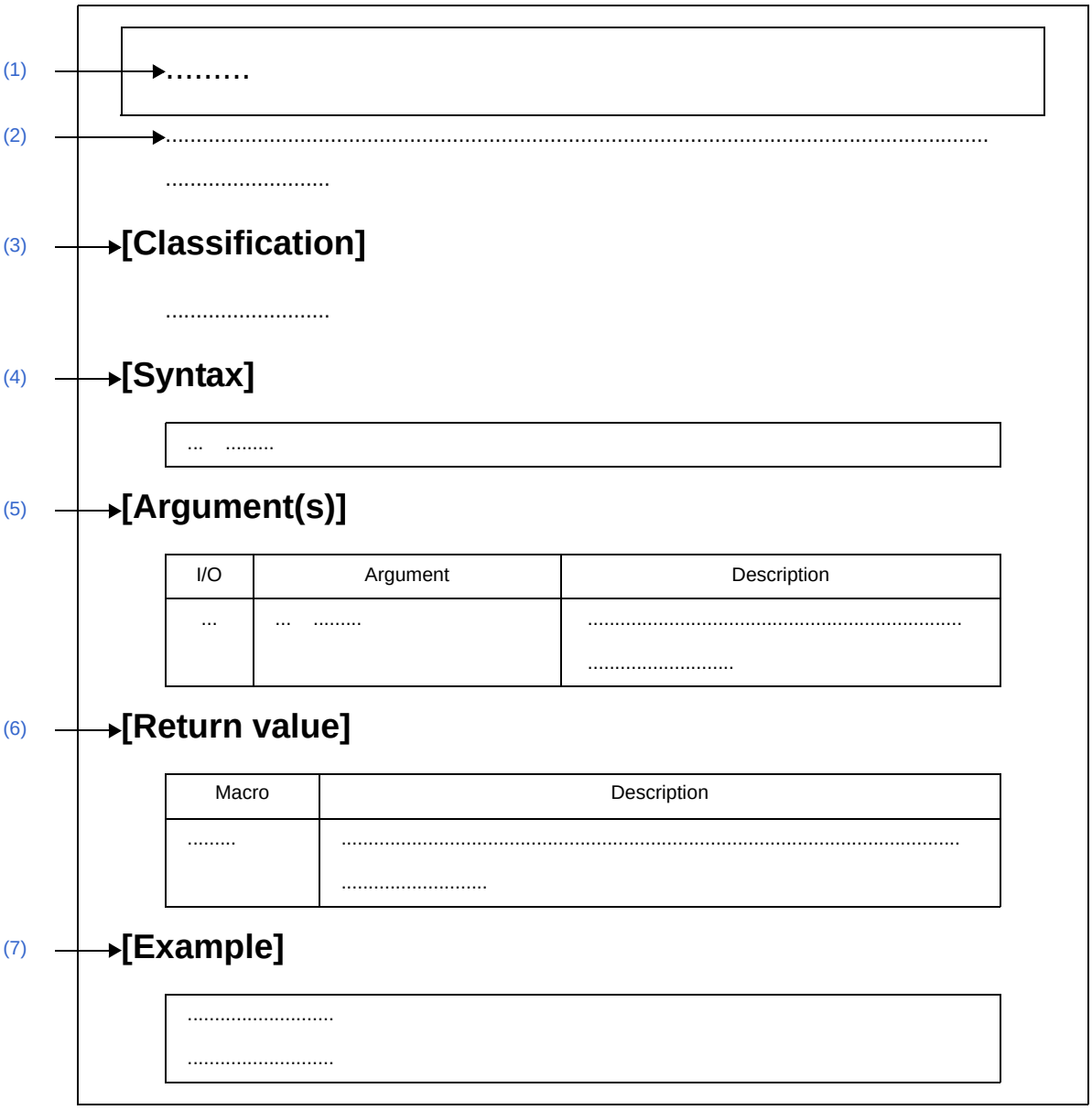
Peripheral Function	API Function Name	Function
Interval Timer	R_IT_Start	Starts the count of the interval timer.
	R_IT_Stop	Ends the count of the interval timer.
	R_IT_Set_PowerOff	Halts the clock supplied to the interval timer.
Comparator	R_COMP_Create	Performs initialization necessary to control the comparator functions.
	R_COMP_Create_UserInit	Performs user-defined initialization relating to the comparator.
	R_COMPn_Interrupt	Performs processing in response to the comparator interrupt INTCMP*n*.
	R_COMPn_Start	Begins comparison of reference input voltage and analog input voltage.
	R_COMPn_Stop	Stops comparison of reference input voltage and analog input voltage.
Clock Output/ Buzzer Output	R_PCLBUZn_Create	Performs initialization necessary to control clock/buzzer output control circuit functions.
	R_PCLBUZn_Create_UserInit	Performs user-defined initialization relating to the clock/ buzzer output control circuits.
	R_PCLBUZn_Start	Starts clock/buzzer output.
	R_PCLBUZn_Stop	Ends clock/buzzer output.
Data Transfer Controller	R_DTC_Create	Performs initialization necessary to control the data transfer controller functions.
	R_DTC_Create_UserInit	Performs user-defined initialization relating to the data transfer controller.
	R_DTCn_Start	Enables operation of the data transfer controller.
	R_DTCn_Stop	Disables operation of the data transfer controller.
	R_DTC_Set_PowerOff	Halts the clock supplied to the data transfer controller.
Event Link Controller	R_ELC_Create	Performs initialization necessary to control the event link controller functions.
	R_ELC_Create_UserInit	Performs user-defined initialization relating to the event link controller.
	R_ELC_Stop	Disables operation of the event link controller.
DMA Controller	R_DMACn_Create	Performs initialization necessary to control DMA controller functions.
	R_DMACn_Create_UserInit	Performs user-defined initialization relating to the DMA controller.
	R_DMACn_Interrupt	This API function is called as the interrupt process corresponding to the interval interrupt INTWDTI.
	R_DMACn_Start	Enables operation of channel *n*.
	R_DMACn_Stop	Disables operation of channel *n*.
	R_DMACn_SoftwareTriggerOn	Starts DMA transfer when DMA operation is enabled.
Voltage Detector	R_LVD_Create	Performs initialization necessary to control voltage detector functions.

Peripheral Function	API Function Name	Function
Voltage Detector	R_LVD_Create_UserInit	Performs user-defined initialization relating to the voltage detector.
	R_LVD_Interrupt	Performs processing in response to the voltage detection interrupt INTLVI.
	R_LVD_InterruptMode_Start	Starts voltage detection (when in interrupt mode, and interrupt & reset mode).

C.3 Function Reference

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

Figure C-1. Notation Format of API Functions



(1) Name

Indicates the name of the API function.

(2) Outline

Outlines the functions of the API function.

(3) [Classification]

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

(4) [Syntax]

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

(5) [Argument(s)]

API function arguments are explained in the following format.

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

(a) I/O

Argument classification

I ... Input argument

O ... Output argument

(b) Argument

Argument data type

(c) Description

Description of argument

(6) [Return value]

API function return value is explained in the following format.

Macro	Description
(a)	(b)

(a) Macro

Macro of return value

(b) Description

Description of return value

(7) [Example]

Shows an example of the API function in use.

C.3.1 Clock Generator

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

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

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

R_CGC_Create

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

[Classification]

r_cgc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_CGC_Create_UserInit

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

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

[Classification]

r_cgc_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_CGC_Get_ResetSource

Performs processing in response to RESET signal.

[Classification]

r_cgc_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_CGC_Set_ClockMode

Changes the CPU clock/peripheral hardware clock.

[Classification]

r_cgc.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum ClockMode mode;`	Clock generator type HIOCLK: High-speed onchip oscillator SYSX1CLK: X1 clock SYSEXTCLK: External main system clock

[Return value]

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

R_CGC_Set_CRCOn

Starts the CRC operation function.

[Classification]

r_cgc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

C.3.2 Port

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

Table C-3. API Functions: [Port]

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

R_PORT_Create

Performs initialization necessary to control port functions.

[Classification]

r_port.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_PORT_Create_UserInit

Performs user-defined initialization relating to the port.

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

[Classification]

r_port_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

C.3.3 Interrupt

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

Table C-4. API Functions: [Interrupt]

API Function Name	Function
R_INTC_Create	Performs initialization necessary to control the external maskable interrupt INTP*n* functions.
R_INTC_Create_UserInit	Performs user-defined initialization relating to the external maskable interrupt INTP*n* functions.
R_INTCn_Interrupt	Performs processing in response to the external maskable interrupt INTP*n*.
R_INTCn_Start	Enables the acceptance of the external maskable interrupts INTP*n*.
R_INTCn_Stop	Disables the acceptance of the external maskable interrupts INTP*n*.
R_KEY_Create	Performs initialization necessary to control the key interrupt INTKR functions.
R_KEY_Create_UserInit	Performs user-defined initialization relating to the key interrupt INTKR functions.
R_KEY_Interrupt	Performs processing in response to the key interrupt INTKR.
R_KEY_Start	Enables the acceptance of the key interrupts INTKR.
R_KEY_Stop	Disables the acceptance of the key interrupts INTKR.

R_INTC_Create

Performs initialization necessary to control the external maskable interrupt INTP*n* functions.

[Classification]

r_intc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_INTC_Create_UserInit

Performs user-defined initialization relating to the external maskable interrupt INTP*n* functions.

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

[Classification]

r_intc_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_INTC*n*_Interrupt

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

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

[Classification]

r_intc_user.c

[Syntax]

```
void    R_INTCn_Interrupt ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTC*n*_Start

Enables the acceptance of the external maskable interrupts INTP*n*.

[Classification]

r_intc.c

[Syntax]

```
void    R_INTCn_Start ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTC*n*_Stop

Disables the acceptance of the external maskable interrupts INTP*n*.

[Classification]

r_intc.c

[Syntax]

```
void    R_INTCn_Stop ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_KEY_Create

Performs initialization necessary to control the key interrupt INTKR functions.

[Classification]

r_intc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_KEY_Create_UserInit

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

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

[Classification]

r_intc_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_KEY_Interrupt

Performs processing in response to the key interrupt INTKR.

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

[Classification]

r_intc_user.c

[Syntax]

```
void    R_KEY_Interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_KEY_Start

Enables the acceptance of the key interrupts INTKR.

[Classification]

r_intc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_KEY_Stop

Disables the acceptance of the key interrupts INTKR.

[Classification]

r_intc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

C.3.4 Serial

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

Table C-5. API Functions: [Serial]

API Function Name	Function
R_SAUm_Create	Performs initialization necessary to control the serial array unit and serial interface functions.
R_SAUm_Create_UserInit	Performs user-defined initialization related to the serial array unit and serial interface functions.
R_SAUm_Set_PowerOff	Halts the clock supplied to the serial array unit.
R_SAU0_Set_SnoozeOn	Enables the switch from STOP mode to SNOOZE mode.
R_SAU0_Set_SnoozeOff	Disables the switch from STOP mode to SNOOZE mode.
R_UARTn_Create	Performs initialization of the serial interface (UART) channel.
R_UARTn_Interrupt_Send	Performs processing in response to the UART transmission end interrupt INTST*n*.
R_UARTn_Interrupt_Receive	Performs processing in response to the UART reception end interrupt INTSR*n*.
R_UARTn_Interrupt_Error	Performs processing in response to the reception error interrupt INTSRE*n*.
R_UARTn_Start	Sets UART communication to standby mode.
R_UARTn_Stop	Ends UART communication.
R_UARTn_Send	Starts UART data transmission.
R_UARTn_Receive	Starts UART data reception.
R_UARTn_Callback_SendEnd	Performs processing in response to the UART transmission end interrupt INTST*n*.
R_UARTn_Callback_ReceiveEnd	Performs processing in response to the UART reception end interrupt INTSR*n*.
R_UARTn_Callback_Error	Performs processing in response to the UART reception error interrupt INTSRE*n*.
R_UARTn_Callback_SoftwareOverRun	Performs processing in response to detection of overrun error.
R_CSImn_Create	Performs initialization of the serial interface (CSI) channel.
R_CSImn_Interrupt	Performs processing in response to the CSI communication end interrupt INTCSI*mn*.
R_CSImn_Start	Sets CSI communication to standby mode.
R_CSImn_Stop	Ends CSI communication.
R_CSImn_Send	Starts CSI data transmission.
R_CSImn_Receive	Starts CSI data reception.
R_CSImn_Send_Receive	Starts CSI data transmission/reception.
R_CSImn_Callback_SendEnd	Performs processing in response to the CSI transmission end interrupt INTCSI*mn*.
R_CSImn_Callback_ReceiveEnd	Performs processing in response to the CSI reception end interrupt INTCSI*mn*.
R_CSImn_Callback_Error	Performs processing in response to the CSI reception error interrupt INTSRE*n*.
R_IICmn_Create	Performs initialization of the serial interface (simple IIC) channel.
R_IICmn_Interrupt	Performs processing in response to the simple IIC communication end interrupt INTIIC*mn*.

API Function Name	Function
R_IICmn_StartCondition	Generates start conditions.
R_IICmn_StopCondition	Generates stop conditions.
R_IICmn_Stop	Ends simple IIC communication.
R_IICmn_Master_Send	Starts simple IIC master transmission.
R_IICmn_Master_Receive	Starts simple IIC master reception.
R_IICmn_Callback_Master_SendEnd	Performs processing in response to the simple IIC*mn* master transmission end interrupt INTIIC*mn*.
R_IICmn_Callback_Master_ReceiveEnd	Performs processing in response to the simple IIC*mn* master reception end interrupt INTIIC*mn*.
R_IICmn_Callback_Master_Error	Performs processing in response to detection of parity error (ACK error).
R_IICAn_Create	Performs initialization of the serial interface (IICA).
R_IICAn_Create_UserInit	Performs user-defined initialization of the serial interface (IICA).
R_IICAn_Interrupt	Performs processing in response to the IICA communication end interrupt INTIICA*n*.
R_IICAn_StopCondition	Generates stop conditions.
R_IICAn_Stop	Ends IICA communication.
R_IICAn_Set_PowerOff	Halts the clock supplied to the serial interface (IICA).
R_IICAn_Master_Send	Starts IICA master transmission.
R_IICAn_Master_Receive	Starts IICA master reception.
R_IICAn_Callback_Master_SendEnd	Performs processing in response to the IICA master transmission end interrupt INTIICA*n*.
R_IICAn_Callback_Master_ReceiveEnd	Performs processing in response to the IICA master reception end interrupt INTIICA*n*.
R_IICAn_Callback_Master_Error	Performs processing in response to detection of IICA master communication error.
R_IICAn_Slave_Send	Starts IICA slave transmission.
R_IICAn_Slave_Receive	Starts IICA slave reception.
R_IICAn_Callback_Slave_SendEnd	Performs processing in response to the IICA slave transmission end interrupt INTIICA*n*.
R_IICAn_Callback_Slave_ReceiveEnd	Performs processing in response to the IICA slave reception end interrupt INTIICA*n*.
R_IICAn_Callback_Slave_Error	Performs processing in response to detection of IICA slave communication error.
R_IICAn_Callback_GetStopCondition	Performs processing in response to detection of stop condition.

R_SAU*m*_Create

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

[Classification]

r_serial.c

[Syntax]

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

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAU*m*_Create_UserInit

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

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

[Classification]

r_serial_user.c

[Syntax]

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

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAU*m*_Set_PowerOff

Halts the clock supplied to the serial array unit.

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

[Classification]

r_serial.c

[Syntax]

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

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAU0_Set_SnoozeOn

Enables the switch from STOP mode to SNOOZE mode.

[Classification]

r_serial.c

[Syntax]

```
void    R_SAU0_Set_SnoozeOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SAU0_Set_SnoozeOff

Disables the switch from STOP mode to SNOOZE mode.

[Classification]

r_serial.c

[Syntax]

```
void    R_SAU0_Set_SnoozeOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Create

Performs initialization of the serial interface (UART) channel.

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

[Classification]

r_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Interrupt_Send

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

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

[Classification]

r_serial_user.c

[Syntax]

```
void    R_UARTn_Interrupt_Send ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Interrupt_Receive

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

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

[Classification]

r_serial_user.c

[Syntax]

```
void    R_UARTn_Interrupt_Receive ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Interrupt_Error

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

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

[Classification]

r_serial_user.c

[Syntax]

```
void    R_UART n_Interrupt_Error ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Start

Sets UART communication to standby mode.

[Classification]

r_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Stop

Ends UART communication.

[Classification]

r_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Send

Starts UART data transmission.

Remarks 1. This API function repeats the byte-level UART transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

2. When performing a UART transmission, R_UARTn_Start must be called before this API function is called.

[Classification]

r_serial.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_UARTn_Send ( uint8_t *txbuf, uint16_t txnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t *txbuf;`	Pointer to a buffer storing the transmission data
I	`uint16_t txnum;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_UART*n*_Receive

Starts UART data reception.

Remarks 1. This API function performs byte-level UART reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

2. Actual UART reception starts after this API function is called, and R_UARTn_Start is then called.

[Classification]

r_serial.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_UARTn_Receive ( uint8_t *rxbuf, uint16_t rxnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t *rxbuf;`	Pointer to a buffer to store the received data
I	`uint16_t rxnum;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_UART*n*_Callback_SendEnd

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

Remark This API function is called as the callback routine of interrupt process R_UARTn_Interrupt_Send corresponding to the UART transmission end interrupt INTST*n* (performed when number of transmission data specified by R_UARTn_Send parameter *txnum* has been completed).

[Classification]

r_serial_user.c

[Syntax]

```
void    R_UARTn_Callback_SendEnd ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Callback_ReceiveEnd

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

Remark This API function is called as the callback routine of interrupt process R_UARTn_Interrupt_Receive corresponding to the UART reception end interrupt INTSR*n* (performed when number of received data specified by R_UARTn_Receive parameter *rxnum* has been completed).

[Classification]

r_serial_user.c

[Syntax]

```
void    R_UARTn_Callback_ReceiveEnd ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Callback_Error

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

Remark This API function is called as the callback routine of interrupt process R_UARTn_Interrupt_Error corresponding to the reception error interrupt INTSRE*n*.

[Classification]

r_serial_user.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t err_type;`	Trigger for error interrupt 00000*xx*1B: Overrun error 00000*x*1*x*B: Parity error 000001*xx*B: Framing error

[Return value]

None.

R_UART*n*_Callback_SoftwareOverRun

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process R_UARTn_Interrupt_Receive corresponding to the UART reception end interrupt INTSR*n* (process performed when the amount of data received is greater than the parameter *rxnum* specified for R_UARTn_Receive).

[Classification]

r_serial_user.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint16_t rx_data;`	Receive data (greater than the parameter *rxnum* specified for R_UARTn_Receive)

[Return value]

None.

R_CSI*mn*_Create

Performs initialization of the serial interface (CSI) channel.

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

[Classification]

r_serial.c

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_CSI*mn*_Interrupt

Performs processing in response to the CSI communication end interrupt INTCSI*mn*.

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

[Classification]

r_serial_user.c

[Syntax]

```
void    R_CSImn_Interrupt ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_CSI*mn*_Start

Sets CSI communication to standby mode.

[Classification]

r_serial.c

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_CSI*mn*_Stop

Ends CSI communication.

[Classification]

r_serial.c

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_CSI*mn*_Send

Starts CSI data transmission.

Remarks 1. This API function repeats the byte-level CSI transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

2. When performing a CSI transmission, R_CSImn_Start must be called before this API function is called.

[Classification]

r_serial.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_CSImn_Send ( uint8_t *txbuf, uint16_t txnum );
```

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

[Argument(s)]

I/O	Argument	Description
I	`uint8_t *txbuf;`	Pointer to a buffer storing the transmission data
I	`uint16_t txnum;`	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_CSI*mn*_Receive

Starts CSI data reception.

Remarks 1. This API function performs byte-level CSI reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

2. When performing a CSI reception, R_CSImn_Start must be called before this API function is called.

[Classification]

r_serial.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_CSImn_Receive ( uint8_t *rxbuf, uint16_t rxnum );
```

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

[Argument(s)]

I/O	Argument	Description
O	`uint8_t *rxbuf;`	Pointer to a buffer to store the received data
I	`uint16_t rxnum;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_CSI*mn*_Send_Receive

Starts CSI data transmission/reception.

Remarks 1. This API function repeats the byte-level CSI transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

2. This API function performs byte-level CSI reception the number of times specified by the parameter *txnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

3. When performing a CSI reception, R_CSImn_Start must be called before this API function is called.

[Classification]

r_serial.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_CSImn_Send_Receive ( uint8_t *txbuf, uint16_t txnum, uint8_t *rxbuf );
```

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

[Argument(s)]

I/O	Argument	Description
I	`uint8_t *txbuf;`	Pointer to a buffer storing the transmission data
I	`uint16_t txnum;`	Total amount of data to send/receive
O	`uint8_t *rxbuf;`	Pointer to a buffer to store the received data

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_CSI*mn*_Callback_SendEnd

Performs processing in response to the CSI transmission end interrupt INTCSI*mn*.

Remark This API function is called as the callback routine of interrupt process R_CSImn_Interrupt corresponding to the CSI transmission end interrupt INTCSI*mn* (performed when number of transmission data specified by R_CSImn_Send or R_CSImn_Send_Receive parameter *txnum* has been completed).

[Classification]

r_serial_user.c

[Syntax]

```
void    R_CSImn_Callback_SendEnd ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_CSI*mn*_Callback_ReceiveEnd

Performs processing in response to the CSI reception end interrupt INTCSI*mn*.

Remark This API function is called as the callback routine of interrupt process R_CSImn_Interrupt corresponding to the CSI reception end interrupt INTCSI*mn* (performed when number of received data specified by R_CSImn_Receive or R_CSImn_Send_Receive parameter *rxnum* has been completed).

[Classification]

r_serial_user.c

[Syntax]

```
void    R_CSImn_Callback_ReceiveEnd ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_CSI*mn*_Callback_Error

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

Remark This API function is called as the callback routine of interrupt process R_UARTn_Interrupt_Error corresponding to the CSI reception error interrupt INTSRE*n*.

[Classification]

r_serial_user.c

[Syntax]

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

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

[Argument(s)]

I/O	Argument	Description
O	`uint8_t err_type;`	Trigger for error interrupt 00000*xx*1B: Overrun error

[Return value]

None.

R_IIC*mn*_Create

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

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

[Classification]

r_serial.c

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_Interrupt

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

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

[Classification]

r_serial_user.c

[Syntax]

```
void    R_IICmn_Interrupt ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_StartCondition

Generates start conditions.

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

[Classification]

r_serial.c

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_StopCondition

Generates stop conditions.

[Classification]

r_serial.c

[Syntax]

```
void    R_IIC mn _StopCondition ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_Stop

Ends simple IIC communication.

[Classification]

r_serial.c

[Syntax]

```
void    R_IIC mn _Stop ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_Master_Send

Starts simple IIC master transmission.

Remark This API function repeats the byte-level simple IIC master transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

[Classification]

r_serial.c

[Syntax]

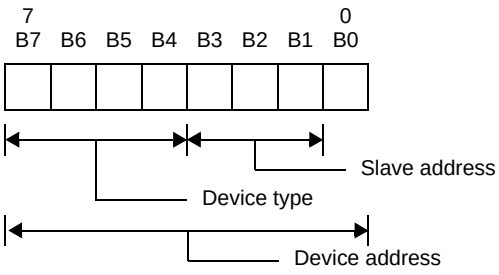
```
#include    "r_cg_macrodriver.h"
void    R_IICmn_Master_Send ( uint8_t adr, uint8_t *txbuf, uint16_t txnum );
```

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

[Argument(s)]

I/O	Argument	Description
I	`uint8_t adr;`	Device address
I	`uint8_t *txbuf;`	Pointer to a buffer storing the transmission data
I	`uint16_t txnum;`	Total amount of data to send

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



[Return value]

None.

R_IIC*mn*_Master_Receive

Starts simple IIC master reception.

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

[Classification]

r_serial.c

[Syntax]

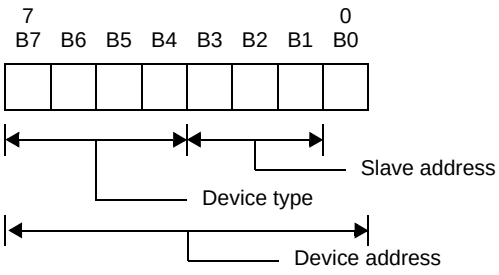
```
#include    "r_cg_macrodriver.h"
void    R_IICmn_Master_Receive ( uint8_t adr, uint8_t *rxbuf, uint16_t rxnum );
```

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

[Argument(s)]

I/O	Argument	Description
I	`uint8_t adr;`	Device address
O	`uint8_t *rxbuf;`	Pointer to a buffer to store the received data
I	`uint16_t rxnum;`	Total amount of data to receive

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



[Return value]

None.

R_IIC*mn*_Callback_Master_SendEnd

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

Remark This API function is called as the callback routine of interrupt process R_IICmn_Interrupt corresponding to the simple IIC*mn* master transmission end interrupt INTIIC*mn* (performed when number of transmission data specified by R_IICmn_Master_Send parameter *txnum* has been completed).

[Classification]

r_serial_user.c

[Syntax]

```
void    R_IICmn_Callback_Master_SendEnd ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_Callback_Master_ReceiveEnd

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

Remark This API function is called as the callback routine of interrupt process R_IICmn_Interrupt corresponding to the simple IIC*mn* master reception end interrupt INTIIC*mn* (performed when number of received data specified by R_IICmn_Master_Receive parameter *rxnum* has been completed).

[Classification]

r_serial_user.c

[Syntax]

```
void    R_IICmn_Callback_Master_ReceiveEnd ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_Callback_Master_Error

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

[Classification]

r_serial_user.c

[Syntax]

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

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

[Argument(s)]

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

[Return value]

None.

R_IICA*n*_Create

Performs initialization of the serial interface (IICA).

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

[Classification]

r_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Create_UserInit

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

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

[Classification]

r_serial_user.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Interrupt

Performs processing in response to the IICA communication end interrupt INTIICA*n*.

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

[Classification]

r_serial_user.c

[Syntax]

```
void    R_IICAn_Interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_StopCondition

Generates stop conditions.

[Classification]

r_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Stop

Ends IICA communication.

[Classification]

r_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Set_PowerOff

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

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

[Classification]

r_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Master_Send

Starts IICA master transmission.

Remark This API function repeats the byte-level IICA master transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

[Classification]

r_serial.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_IICAn_Master_Send ( uint8_t adr, uint8_t *txbuf, uint16_t txnum, uint8_t wait );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t adr;`	Slave address
I	`uint8_t *txbuf;`	Pointer to a buffer storing the transmission data
I	`uint16_t txnum;`	Total amount of data to send
I	`uint8_t wait;`	Setup time of start conditions

[Return value]

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

R_IICA*n*_Master_Receive

Starts IICA master reception.

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

[Classification]

r_serial.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_IICAn_Master_Receive ( uint8_t adr, uint8_t *rxbuf, uint16_t rxnum, uint8_t wait
);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t adr;`	Slave address
O	`uint8_t *rxbuf;`	Pointer to a buffer to store the received data
I	`uint16_t rxnum;`	Total amount of data to receive
I	`uint8_t wait;`	Setup time of start conditions

[Return value]

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

R_IICA*n*_Callback_Master_SendEnd

Performs processing in response to the IICA master transmission end interrupt INTIICA*n*.

Remark This API function is called as the callback routine of interrupt process R_IICAn_Interrupt corresponding to the IICA master transmission end interrupt INTIICA*n*.

[Classification]

r_serial_user.c

[Syntax]

```
void    R_IICAn_Callback_Master_SendEnd ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Callback_Master_ReceiveEnd

Performs processing in response to the IICA master reception end interrupt INTIICA*n*.

Remark This API function is called as the callback routine of interrupt process R_IICAn_Interrupt corresponding to the IICA master reception end interrupt INTIICA*n*.

[Classification]

r_serial_user.c

[Syntax]

```
void    R_IICAn_Callback_Master_ReceiveEnd ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Callback_Master_Error

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

[Classification]

r_serial_user.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

R_IICA*n*_Slave_Send

Starts IICA slave transmission.

Remark This API function repeats the byte-level IICA slave transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

[Classification]

r_serial.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_IICAn_Slave_Send ( uint8_t *txbuf, uint16_t txnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`uint8_t *txbuf;`	Pointer to a buffer storing the transmission data
I	`uint16_t txnum;`	Total amount of data to send

[Return value]

None.

R_IICA*n*_Slave_Receive

Starts IICA slave reception.

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

[Classification]

r_serial.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_IICAn_Slave_Receive ( uint8_t *rxbuf, uint16_t rxnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint8_t *rxbuf;`	Pointer to a buffer to store the received data
I	`uint16_t rxnum;`	Total amount of data to receive

[Return value]

None.

R_IICA*n*_Callback_Slave_SendEnd

Performs processing in response to the IICA slave transmission end interrupt INTIICA*n*.

Remark This API function is called as the callback routine of interrupt process R_IICAn_Interrupt corresponding to the IICA slave transmission end interrupt INTIICA*n*.

[Classification]

r_serial_user.c

[Syntax]

```
void    R_IICAn_Callback_Slave_SendEnd ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Callback_Slave_ReceiveEnd

Performs processing in response to the IICA slave reception end interrupt INTIICA*n*.

Remark This API function is called as the callback routine of interrupt process R_IICAn_Interrupt corresponding to the IICA slave reception end interrupt INTIICA*n*.

[Classification]

r_serial_user.c

[Syntax]

```
void    R_IICAn_Callback_Slave_ReceiveEnd ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Callback_Slave_Error

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

[Classification]

r_serial_user.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

R_IICA*n*_Callback_GetStopCondition

Performs processing in response to detection of stop condition.

[Classification]

r_serial_user.c

[Syntax]

```
void    R_IICAn_Callback_GetStopCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.3.5 A/D Converter

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

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

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

R_ADC_Create

Performs initialization necessary to control A/D converter functions.

[Classification]

r_adc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ADC_Create_UserInit

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

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

[Classification]

r_adc_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ADC_Interrupt

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

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

[Classification]

r_adc_user.c

[Syntax]

```
void    R_ADC_Interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_ComparatorOn

Enables operation of voltage converter.

Remarks 1. About 1 microsecond of stabilization time is required when changing the voltage converter from operation stopped to operation enabled status.
Consequently, about 1 micro second must be left free between the call to this API function and the call to R_ADC_Start.

2. On the [A/D Converter], in the [Comparator operation setting] area, if "Operation" is selected, then the voltage converter will be switched to "always on". There is thus no need to call this API function in this case.

[Classification]

r_adc.c

[Syntax]

```
void    R_ADC_Set_ComparatorOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_ComparatorOff

Disables operation of voltage converter.

[Classification]

r_adc.c

[Syntax]

```
void    R_ADC_Set_ComparatorOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Start

Starts A/D conversion.

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

[Classification]

r_adc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ADC_Stop

Ends A/D conversion.

Remark The voltage converter continues to operate after the process of this API function completes. Consequently, to stop the operation of the voltage converter, you must call R_ADC_Set_ComparatorOff after the process of this API function completes.

[Classification]

r_adc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_PowerOff

Halts the clock supplied to the A/D converter.

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

[Classification]

r_adc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_ADChannel

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

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

[Classification]

r_adc.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum ADChannel channel;`	Analog voltage input pin ADCHANNEL*n*: Input pin

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_ADC_Set_SnoozeOn

Enables the switch from STOP mode to SNOOZE mode.

[Classification]

r_adc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_SnoozeOff

Disables the switch from STOP mode to SNOOZE mode.

[Classification]

r_adc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_TestChannel

Sets the operaiton mode of A/D converter.

[Classification]

r_adc.c

[Syntax]

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

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_ADC_Get_Result

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

[Classification]

r_adc.c

[Syntax]

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

[Argument(s)]

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

[Return value]

None.

R_ADC_Get_Result_8bit

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

[Classification]

r_adc.c

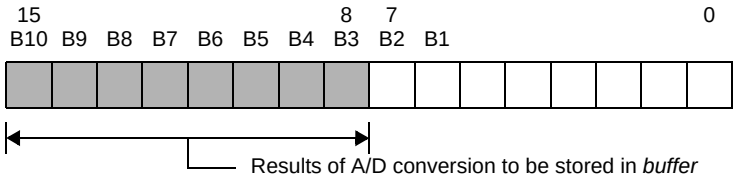
[Syntax]

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

[Argument(s)]

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

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



[Return value]

None.

C.3.6 D/A Converter

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

Sets the analog voltage output to the ANO*n* pin.

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

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

R_DAC_Create

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

[Classification]

r_dac.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_DAC_Create_UserInit

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

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

[Classification]

r_dac_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_DAC*n*_Start

Starts D/A conversion.

[Classification]

r_dac.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DAC*n*_Stop

Ends D/A conversion.

[Classification]

r_dac.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DAC_Set_PowerOff

Halts the clock supplied to the D/A converter.

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

[Classification]

r_dac.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_DAC*n*_Set_ConversionValue

Sets the analog voltage output to the ANO*n* pin.

[Classification]

r_dac.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

C.3.7 Timer

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

Table C-8. API Functions: [Timer]

API Function Name	Function
R_TAUm_Create	Performs initialization necessary to control timer array unit functions.
R_TAUm_Create_UserInit	Performs user-defined initialization relating to the timer array unit.
R_TAUm_Channeln_Interrupt	Performs processing in response to the timer interrupt INTTM*mn*.
R_TAUm_Channeln_Higher8bits_Interrupt	Performs processing in response to the timer interrupt INTTM*mn*H.
R_TAUm_Channeln_Start	Starts the count for channel *n*.
R_TAUm_Channeln_Higher8bits_Start	Starts the count (higher 8-bit) for channel 1 or channel 3.
R_TAUm_Channeln_Lower8bits_Start	Starts the count (lower 8-bit) for channel 1 or channel 3.
R_TAUm_Channeln_Stop	Ends the count for channel *n*.
R_TAUm_Channeln_Higher8bits_Stop	Ends the count (higher 8-bit) for channel 1 or channel 3.
R_TAUm_Channeln_Lower8bits_Stop	Ends the count (lower 8-bit) for channel 1 or channel 3.
R_TAUm_Set_PowerOff	Halts the clock supplied to the timer array unit.
R_TAUm_Channeln_Get_PulseWidth	Captures the high/low-level width measured between pulses of the signal (pulses) input to the TI*mn* pin.
R_TAUm_Channeln_SoftwareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.
R_TMR_RDn_Create	Performs initialization necessary to control the timer RD*n* functions.
R_TMR_RDn_Create_UserInit	Performs user-defined initialization relating to the timer RD*n*.
R_TMR_RDn_Interrupt	Performs processing in response to the timer interrupt.
R_TMR_RDn_Start	Starts the count for timer RD*n*.
R_TMR_RDn_Stop	Ends the count for timer RD*n*.
R_TMR_RDn_Set_PowerOff	Halts the clock supplied to the timer RD*n*.
R_TMR_RDn_Get_PulseWidth	Reads the pulse width of the timer RD*n*.
R_TMR_RGn_Create	Performs initialization necessary to control the timer RG*n* functions.
R_TMR_RGn_Create_UserInit	Performs user-defined initialization relating to the timer RG*n*.
R_TMR_RGn_Interrupt	Performs processing in response to the timer interrupt.
R_TMR_RGn_Start	Starts the count for timer RG*n*.
R_TMR_RGn_Stop	Ends the count for timer RG*n*.
R_TMR_RGn_Set_PowerOff	Halts the clock supplied to the timer RG*n*.
R_TMR_RGn_Get_PulseWidth	Reads the pulse width of the timer RG*n*.
R_TMR_RJ0_Create	Performs initialization necessary to control the timer RJ0 functions.
R_TMR_RJ0_Create_UserInit	Performs user-defined initialization relating to the timer RJ0.
R_TMR_RJ0_Interrupt	Performs processing in response to the timer interrupt.
R_TMR_RJ0_Start	Starts the count for timer RJ0.
R_TMR_RJ0_Stop	Ends the count for timer RJ0.
R_TMR_RJ0_Set_PowerOff	Halts the clock supplied to the timer RJ0.

API Function Name	Function
R_TMR_RJ0_Get_PulseWidth	Reads the pulse width of the timer RJ0.

R_TAU*m*_Create

Performs initialization necessary to control timer array unit functions.

[Classification]

r_timer.c

[Syntax]

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

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Create_UserInit

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

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

[Classification]

r_timer_user.c

[Syntax]

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

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Interrupt

Performs processing in response to the timer interrupt INTTM*mn*.

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

[Classification]

r_timer_user.c

[Syntax]

```
void    R_TAUm_Channeln_Interrupt ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Higher8bits_Interrupt

Performs processing in response to the timer interrupt INTTM*mn*H.

Remark This API function is called as the interrupt process corresponding to the timer interrupt INTTM*mn*H.

[Classification]

r_timer_user.c

[Syntax]

```
void    R_TAUm_Channeln_Higher8bits_Interrupt ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Start

Starts the count for channel *n*.

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

[Classification]

r_timer.c

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Higher8bits_Start

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

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

[Classification]

r_timer.c

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Lower8bits_Start

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

Remarks 1. This API function can only be called when the timer array unit is used as a 8-bit timer.

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

[Classification]

r_timer.c

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Stop

Ends the count for channel *n*.

[Classification]

r_timer.c

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Higher8bits_Stop

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

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

[Classification]

r_timer.c

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Lower8bits_Stop

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

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

[Classification]

r_timer.c

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Set_PowerOff

Halts the clock supplied to the timer array unit.

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

[Classification]

r_timer.c

[Syntax]

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

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Get_PulseWidth

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

[Classification]

r_timer.c

[Syntax]

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

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

[Argument(s)]

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

[Return value]

None.

R_TAU*m*_Channel*n*_SoftwareTriggerOn

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

[Classification]

r_timer.c

[Syntax]

```
void    R_TAUm_Channeln_SoftwareTriggerOn ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Create

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

[Classification]

r_timer.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Create_UserInit

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

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

[Classification]

r_timer_user.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Interrupt

Performs processing in response to the timer interrupt.

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

[Classification]

r_timer_user.c

[Syntax]

```
void    R_TMR_RDn_Interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Start

Starts the count for timer RD*n*.

[Classification]

r_timer.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Stop

Ends the count for timer RD*n*.

[Classification]

r_timer.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Set_PowerOff

Halts the clock supplied to the timer RD*n*.

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

[Classification]

r_timer.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Get_PulseWidth

Reads the pulse width of the timer RD*n*.

Remarks 1. This API function can only be called when the timer RD*n* is being used for input capture function.

2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Classification]

r_timer.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_timer.h"
MD_STATUS   R_TMR_RDn_Get_PulseWidth ( uint32_t *activewidth, uint32_t *inactivewidth, enum TMChannel channel );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint32_t *activewidth;`	Pointer to an area storing the active level width that was read
O	`uint32_t *inactivewidth;`	Pointer to an area storing the inactive level width that was read
I	`enum TMChannel channel;`	Pin to read TMCHANNELA: TRDIOA*n* pin TMCHANNELB: TRDIOB*n* pin TMCHANNELC: TRDIOC*n* pin TMCHANNELD: TRDIOD*n* pin

[Return value]

Macro	Description
MD_OK	Normal completion

R_TMR_RG*n*_Create

Performs initialization necessary to control the timer RG*n* functions.

[Classification]

r_timer.c

[Syntax]

```
void    R_TMR_RGn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG*n*_Create_UserInit

Performs user-defined initialization relating to the timer RG*n*.

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

[Classification]

r_timer_user.c

[Syntax]

```
void    R_TMR_RGn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG*n*_Interrupt

Performs processing in response to the timer interrupt.

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

[Classification]

r_timer_user.c

[Syntax]

```
void    R_TMR_RGn_Interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG*n*_Start

Starts the count for timer RG*n*.

[Classification]

r_timer.c

[Syntax]

```
void    R_TMR_RGn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG*n*_Stop

Ends the count for timer RG*n*.

[Classification]

r_timer.c

[Syntax]

```
void    R_TMR_RGn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG*n*_Set_PowerOff

Halts the clock supplied to the timer RG*n*.

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

[Classification]

r_timer.c

[Syntax]

```
void    R_TMR_RGn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG*n*_Get_PulseWidth

Reads the pulse width of the timer RG*n*.

Remarks 1. This API function can only be called when the timer RG*n* is being used for input capture function.

2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Classification]

r_timer.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_timer.h"
MD_STATUS   R_TMR_RGn_Get_PulseWidth ( uint32_t *activewidth, uint32_t *inactivewidth, enum TMChannel channel );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`uint32_t *activewidth;`	Pointer to an area storing the active level width that was read from the TRGIOA pin
O	`uint32_t *inactivewidth;`	Pointer to an area storing the inactive level width that was read from the TRGIOA pin
I	`enum TMChannel channel;`	Pin to read TMCHANNELA: TRGIOA*n* pin TMCHANNELB: TRGIOB*n* pin

[Return value]

Macro	Description
MD_OK	Normal completion

R_TMR_RJ0_Create

Performs initialization necessary to control the timer RJ0 functions.

[Classification]

r_timer.c

[Syntax]

```
void    R_TMR_RJ0_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Create_UserInit

Performs user-defined initialization relating to the timer RJ0.

Remark This API function is called as the R_TMR_RJ0_Create callback routine.

[Classification]

r_timer_user.c

[Syntax]

```
void    R_TMR_RJ0_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Interrupt

Performs processing in response to the timer interrupt.

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

[Classification]

r_timer_user.c

[Syntax]

```
void    R_TMR_RJ0_Interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Start

Starts the count for timer RJ0.

[Classification]

r_timer.c

[Syntax]

```
void    R_TMR_RJ0_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Stop

Ends the count for timer RJ0.

[Classification]

r_timer.c

[Syntax]

```
void    R_TMR_RJ0_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Set_PowerOff

Halts the clock supplied to the timer RJ0.

Remark Calling this API function changes the timer RJ0 to reset status. For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_timer.c

[Syntax]

```
void    R_TMR_RJ0_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ0_Get_PulseWidth

Reads the pulse width of the timer RJ0.

Remarks 1. This API function can only be called when the timer RJ0 is being used for pulse width measurement mode / pulse period measurement mode.

2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Classification]

r_timer.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
void    R_TMR_RJ0_Get_PulseWidth ( uint32_t *activewidth );
```

[Argument(s)]

I/O	Argument	Description
O	`uint32_t *activewidth;`	Pointer to an area storing the active level width that was read from the TRJ0IO pin

[Return value]

None.

C.3.8 Watchdog Timer

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

Table C-9. API Functions: [Watchdog Timer]

API Function Name	Function
R_WDT_Create	Performs initialization necessary to control watchdog timer functions.
R_WDT_Create_UserInit	Performs user-defined initialization relating to the watchdog timer.
R_WDT_Interrupt	Performs processing in response to the interval interrupt INTWDTI of watchdog timer.
R_WDT_Restart	Clears the watchdog timer counter and resumes counting.

R_WDT_Create

Performs initialization necessary to control watchdog timer functions.

[Classification]

r_wdt.c

[Syntax]

```
void    R_WDT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WDT_Create_UserInit

Performs user-defined initialization relating to the watchdog timer.

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

[Classification]

r_wdt_user.c

[Syntax]

```
void    R_WDT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WDT_Interrupt

Performs processing in response to the interval interrupt INTWDTI of watchdog timer.

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

[Classification]

r_wdt_user.c

[Syntax]

```
void    R_WDT_Interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WDT_Restart

Clears the watchdog timer counter and resumes counting.

[Classification]

r_wdt.c

[Syntax]

```
void    R_WDT_Restart ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.9 Real-time Clock

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

Table C-10. API Functions: [Real-time Clock]

API Function Name	Function
R_RTC_Create	Performs initialization necessary to control real-time clock functions.
R_RTC_Create_UserInit	Performs user-defined initialization relating to the real-time clock.
R_RTC_Interrupt	Performs processing in response to the real-time clock interrupt INTRTC.
R_RTC_Start	Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second).
R_RTC_Stop	Ends the count of the real-time clock (year, month, weekday, day, hour, minute, second).
R_RTC_Set_PowerOff	Halts the clock supplied to the real-time clock.
R_RTC_Set_HourSystem	Sets the clock type (12-hour or 24-hour clock) of the real-time clock.
R_RTC_Set_CounterValue	Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.
R_RTC_Get_CounterValue	Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.
R_RTC_Set_ConstPeriodInterruptOn	Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.
R_RTC_Set_ConstPeriodInterruptOff	Ends the cyclic interrupt function.
R_RTC_Callback_ConstPeriod	Performs processing in response to the cyclic interrupt INTRTC.
R_RTC_Set_AlarmOn	Starts the alarm interrupt function.
R_RTC_Set_AlarmOff	Ends the alarm interrupt function.
R_RTC_Set_AlarmValue	Sets the alarm conditions (weekday, hour, minute).
R_RTC_Get_AlarmValue	Reads the alarm conditions (weekday, hour, minute).
R_RTC_Callback_Alarm	Performs processing in response to the alarm interrupt INTRTC.
R_RTC_Set_RTC1HZOn	Enables output of the correction clock (1 Hz) to the RTC1HZ pin.
R_RTC_Set_RTC1HZOff	Disables output of the correction clock (1 Hz) to the RTC1HZ pin.

R_RTC_Create

Performs initialization necessary to control real-time clock functions.

[Classification]

r_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_RTC_Create_UserInit

Performs user-defined initialization relating to the real-time clock.

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

[Classification]

r_rtc_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_RTC_Interrupt

Performs processing in response to the real-time clock interrupt INTRTC.

Remark This API function is called as the interrupt process corresponding to the real-time clock interrupt INTRTC.

[Classification]

r_rtc_user.c

[Syntax]

```
void    R_RTC_Interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Start

Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second).

[Classification]

r_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_RTC_Stop

Ends the count of the real-time clock (year, month, weekday, day, hour, minute, second).

[Classification]

r_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_PowerOff

Halts the clock supplied to the real-time clock.

Remark Calling this API function changes the real-time clock to reset status. For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_rtc.c

[Syntax]

```
void    R_RTC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_HourSystem

Sets the clock type (12-hour or 24-hour clock) of the real-time clock.

[Classification]

r_rtc.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
MD_STATUS   R_RTC_Set_HourSystem ( enum RTCHourSystem hoursystem );
```

[Argument(s)]

I/O	Argument	Description
I	`enum RTCHourSystem hoursystem;`	Clock type HOUR12: 12-hour clock HOUR24: 24-hour clock

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)
MD_BUSY2	Stopping count process (after change to setting)
MD_ARGERROR	Invalid argument specification

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Set_CounterValue

Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.

[Classification]

r_rtc.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
MD_STATUS   R_RTC_Set_CounterValue ( struct RTCCounterValue counterwriteval );
```

[Argument(s)]

I/O	Argument	Description
I	`struct RTCCounterValue counterwriteval;`	Counter value

Remark Below is an example of the structure RTCCounterValue (counter value) for the real-time clock.

```
struct  RTCCounterValue {
    uint8_t Sec;    /* second */
    uint8_t Min;    /* Minute */
    uint8_t Hour;   /* Hour */
    uint8_t Day;    /* Day */
    uint8_t Week;   /* Weekday (0: Sunday, 6: Saturday) */
    uint8_t Month;  /* Month */
    uint8_t Year;   /* Year */
};
```

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)
MD_BUSY2	Stopping count process (after change to setting)

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Get_CounterValue

Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.

[Classification]

r_rtc.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
MD_STATUS   R_RTC_Get_CounterValue ( struct RTCCounterValue *counterreadval );
```

[Argument(s)]

I/O	Argument	Description
O	`struct RTCCounterValue *counterreadval;`	Pointer to structure in which to store the counter value being read

Remark See R_RTC_Set_CounterValue for details about the RTCCounterValue counter value.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before reading)
MD_BUSY2	Stopping count process (after reading)

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Set_ConstPeriodInterruptOn

Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.

[Classification]

r_rtc.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
MD_STATUS   R_RTC_Set_ConstPeriodInterruptOn ( enum RTCINTPeriod period );
```

[Argument(s)]

I/O	Argument	Description
I	`enum RTCINTPeriod period;`	Interrupt INTRTC cycle HALFSEC: 0.5 seconds ONESEC: 1 second ONEMIN: 1 minute ONEHOUR: 1 hour ONEDAY: 1 day ONEMONTH: 1 month

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_RTC_Set_ConstPeriodInterruptOff

Ends the cyclic interrupt function.

[Classification]

r_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_RTC_Callback_ConstPeriod

Performs processing in response to the cyclic interrupt INTRTC.

Remark This API function is called as the callback routine of interrupt process R_RTC_Interrupt corresponding to the cyclic interrupt INTRTC.

[Classification]

r_rtc_user.c

[Syntax]

```
void    R_RTC_Callback_ConstPeriod ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_AlarmOn

Starts the alarm interrupt function.

[Classification]

r_rtc.c

[Syntax]

```
void    R_RTC_Set_AlarmOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_AlarmOff

Ends the alarm interrupt function.

[Classification]

r_rtc.c

[Syntax]

```
void    R_RTC_Set_AlarmOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_AlarmValue

Sets the alarm conditions (weekday, hour, minute).

[Classification]

r_rtc.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
void    R_RTC_Set_AlarmValue ( struct RTCAlarmValue alarmval );
```

[Argument(s)]

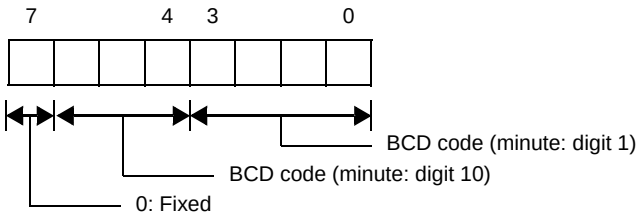
I/O	Argument	Description
I	`struct RTCAlarmValue alarmval;`	Alarm conditions (weekday, hour, minute)

Remark Below is shown the structure RTCAlarmValue (alarm conditions).

```
struct  RTCAlarmValue {
    uint8_t Alarmwm;    /* Minute */
    uint8_t Alarmwh;    /* Hour */
    uint8_t Alarmww;    /* Weekday */
};
```

- Alarmwm (Minute)

  Below are shown the meanings of each bit of the structure member Alarmwm.

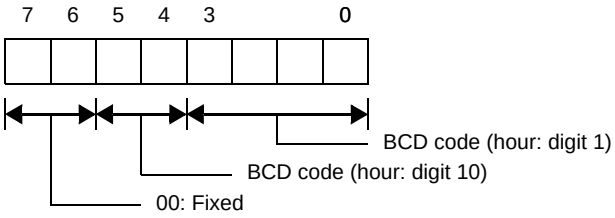


- Alarmwh (Hour)

  Below are shown the meanings of each bit of the structure member Alarmwh. If the real-time clock is set to the 12-hour clock, then bit 5 has the following meaning.

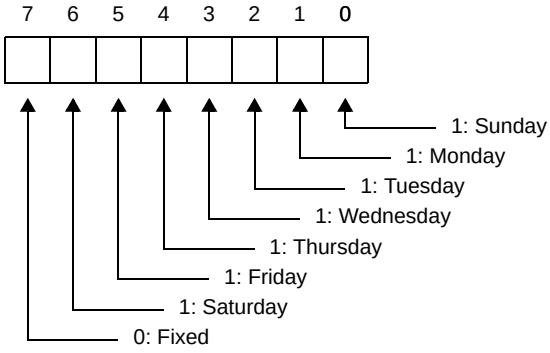
  0: AM

  1: PM



- Alarmww (Weekday)

  Below are shown the meanings of each bit of the structure member Alarmww.



[Return value]

None.

R_RTC_Get_AlarmValue

Reads the alarm conditions (weekday, hour, minute).

[Classification]

r_rtc.c

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
void    R_RTC_Get_AlarmValue ( struct RTCAlarmValue *alarmval );
```

Remark See R_RTC_Set_AlarmValue for details about RTCAlarmValue (alarm conditions).

[Argument(s)]

I/O	Argument	Description
O	`struct RTCAlarmValue *alarmval;`	Pointer to structure in which to store the conditions being read

[Return value]

None.

R_RTC_Callback_Alarm

Performs processing in response to the alarm interrupt INTRTC.

Remark This API function is called as the callback routine of interrupt process R_RTC_Interrupt corresponding to the alarm interrupt INTRTC.

[Classification]

r_rtc_user.c

[Syntax]

```
void    R_RTC_Callback_Alarm ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTC1HZOn

Enables output of the correction clock (1 Hz) to the RTC1HZ pin.

[Classification]

r_rtc.c

[Syntax]

```
void    R_RTC_Set_RTC1HZOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTC1HZOff

Disables output of the correction clock (1 Hz) to the RTC1HZ pin.

[Classification]

r_rtc.c

[Syntax]

```
void    R_RTC_Set_RTC1HZOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.10 Interval Timer

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

Table C-11. API Functions: [Interval Timer]

API Function Name	Function
R_IT_Create	Performs initialization necessary to control interval timer functions.
R_IT_Create_UserInit	Performs user-defined initialization relating to the interval timer.
R_IT_Interrupt	Performs processing in response to the interval timer interrupt INTIT.
R_IT_Start	Starts the count of the interval timer.
R_IT_Stop	Ends the count of the interval timer.
R_IT_Set_PowerOff	Halts the clock supplied to the interval timer.

R_IT_Create

Performs initialization necessary to control interval timer functions.

[Classification]

r_it.c

[Syntax]

```
void    R_IT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Create_UserInit

Performs user-defined initialization relating to the interval timer.

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

[Classification]

r_it_user.c

[Syntax]

```
void    R_IT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Interrupt

Performs processing in response to the interval timer interrupt INTIT.

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

[Classification]

r_it_user.c

[Syntax]

```
void    R_IT_Interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Start

Starts the count of the interval timer.

[Classification]

r_it.c

[Syntax]

```
void    R_IT_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Stop

Ends the count of the interval timer.

[Classification]

r_it.c

[Syntax]

```
void    R_IT_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Set_PowerOff

Halts the clock supplied to the interval timer.

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

[Classification]

r_it.c

[Syntax]

```
void    R_IT_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.11 Comparator

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

Table C-12. API Functions: [Comparator]

API Function Name	Function
R_COMP_Create	Performs initialization necessary to control the comparator functions.
R_COMP_Create_UserInit	Performs user-defined initialization relating to the comparator.
R_COMPn_Interrupt	Performs processing in response to the comparator interrupt INTCMP*n*.
R_COMPn_Start	Begins comparison of reference input voltage and analog input voltage.
R_COMPn_Stop	Stops comparison of reference input voltage and analog input voltage.

R_COMP_Create

Performs initialization necessary to control the comparator functions.

[Classification]

r_comp.c

[Syntax]

```
void    R_COMP_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_COMP_Create_UserInit

Performs user-defined initialization relating to the comparator.

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

[Classification]

r_comp_user.c

[Syntax]

```
void    R_COMP_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_COMP*n*_Interrupt

Performs processing in response to the comparator interrupt INTCMP*n*.

Remark This API function is called as the interrupt process corresponding to the comparator interrupt INTCMP*n*.

[Classification]

r_comp_user.c

[Syntax]

```
void    R_COMPn_Interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP*n*_Start

Begins comparison of reference input voltage and analog input voltage.

[Classification]

r_comp.c

[Syntax]

```
void    R_COMPn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP*n*_Stop

Stops comparison of reference input voltage and analog input voltage.

[Classification]

r_comp.c

[Syntax]

```
void    R_COMPn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.3.12 Clock Output/Buzzer Output

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

Table C-13. API Functions: [Clock Output/Buzzer Output]

API Function Name	Function
R_PCLBUZn_Create	Performs initialization necessary to control clock/buzzer output control circuit functions.
R_PCLBUZn_Create_UserInit	Performs user-defined initialization relating to the clock/buzzer output control circuits.
R_PCLBUZn_Start	Starts clock/buzzer output.
R_PCLBUZn_Stop	Ends clock/buzzer output.

R_PCLBUZ*n*_Create

Performs initialization necessary to control clock/buzzer output control circuit functions.

[Classification]

r_pclbuz.c

[Syntax]

```
void    R_PCLBUZn_Create ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ*n*_Create_UserInit

Performs user-defined initialization relating to the clock/buzzer output control circuits.

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

[Classification]

r_pclbuz_user.c

[Syntax]

```
void    R_PCLBUZn_Create_UserInit ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ*n*_Start

Starts clock/buzzer output.

[Classification]

r_pclbuz.c

[Syntax]

```
void    R_PCLBUZn_Start ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ*n*_Stop

Ends clock/buzzer output.

[Classification]

r_pclbuz.c

[Syntax]

```
void    R_PCLBUZn_Stop ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

C.3.13 Data Transfer Controller

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

Table C-14. API Functions: [Data Transfer Controller]

API Function Name	Function
R_DTC_Create	Performs initialization necessary to control the data transfer controller functions.
R_DTC_Create_UserInit	Performs user-defined initialization relating to the data transfer controller.
R_DTCn_Start	Enables operation of the data transfer controller.
R_DTCn_Stop	Disables operation of the data transfer controller.
R_DTC_Set_PowerOff	Halts the clock supplied to the data transfer controller.

R_DTC_Create

Performs initialization necessary to control the data transfer controller functions.

[Classification]

r_dtc.c

[Syntax]

```
void    R_DTC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTC_Create_UserInit

Performs user-defined initialization relating to the data transfer controller.

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

[Classification]

r_dtc_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_DTC*n*_Start

Enables operation of the data transfer controller.

[Classification]

r_dtc.c

[Syntax]

```
void    R_DTCn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTC*n*_Stop

Disables operation of the data transfer controller.

[Classification]

r_dtc.c

[Syntax]

```
void    R_DTCn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTC_Set_PowerOff

Halts the clock supplied to the data transfer controller.

Remark Calling this API function changes the data transfer controller to reset status. For this reason, writes to the control registers after this API function is called are ignored.

[Classification]

r_dtc.c

[Syntax]

```
void    R_DTC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.14 Event Link Controller

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

Table C-15. API Functions: [Event Link Controller]

API Function Name	Function
R_ELC_Create	Performs initialization necessary to control the event link controller functions.
R_ELC_Create_UserInit	Performs user-defined initialization relating to the event link controller.
R_ELC_Stop	Disables operation of the event link controller.

R_ELC_Create

Performs initialization necessary to control the event link controller functions.

[Classification]

r_elc.c

[Syntax]

```
void    R_ELC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_Create_UserInit

Performs user-defined initialization relating to the event link controller.

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

[Classification]

r_elc_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_ELC_Stop

Disables operation of the event link controller.

[Classification]

r_elc.c

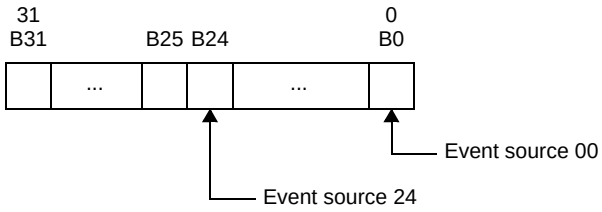
[Syntax]

```
void    R_ELC_Stop ( uint32_t event );
```

[Argument(s)]

I/O	Argument	Description
I	`uint32_t event;`	Disabled event source

Remark Below is shown the format for specifying disabled event source *event*.
In case of setting the *event* to 0x01010101, the event link operations of event source 00, 08, 16, 24 are prohibited.



[Return value]

None.

C.3.15 DMA Controller

Below is a list of API functions output by the Code Generator for DMA (Direct Memory Access) controller use.

Table C-16. API Functions: [DMA Controller]

API Function Name	Function
R_DMACn_Create	Performs initialization necessary to control DMA controller functions.
R_DMACn_Create_UserInit	Performs user-defined initialization relating to the DMA controller.
R_DMACn_Interrupt	Performs processing in response to the DMA transfer end interrupt INTDMA*n*.
R_DMACn_Start	Enables operation of channel *n*.
R_DMACn_Stop	Disables operation of channel *n*.
R_DMACn_SoftwareTriggerOn	Starts DMA transfer when DMA operation is enabled.

R_DMAC*n*_Create

Performs initialization necessary to control DMA controller functions.

[Classification]

r_dmac.c

[Syntax]

```
void    R_DMACn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Create_UserInit

Performs user-defined initialization relating to the DMA controller.

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

[Classification]

r_dmac_user.c

[Syntax]

```
void    R_DMACn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Interrupt

Performs processing in response to the DMA transfer end interrupt INTDMA*n*.

Remark This API function is called as the interrupt process corresponding to the DMA transfer end interrupt INTDMA*n*.

[Classification]

r_dmac_user.c

[Syntax]

```
void    R_DMACn_Interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Start

Enables operation of channel *n*.

[Classification]

r_dmac.c

[Syntax]

```
void    R_DMACn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Stop

Disables operation of channel *n*.

Remarks 1. This API function does not forcibly terminate DMA transfer.

2. Before using this API function, you must confirm that transmission has ended.

[Classification]

r_dmac.c

[Syntax]

```
void    R_DMACn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_SoftwareTriggerOn

Starts DMA transfer when DMA operation is enabled.

[Classification]

r_dmac.c

[Syntax]

```
void    R_DMACn_SoftwareTriggerOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.3.16 Voltage Detector

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

Table C-17. API Functions: [Voltage Detector]

API Function Name	Function
R_LVD_Create	Performs initialization necessary to control voltage detector functions.
R_LVD_Create_UserInit	Performs user-defined initialization relating to the voltage detector.
R_LVD_Interrupt	Performs processing in response to the voltage detection interrupt INTLVI.
R_LVD_InterruptMode_Start	Starts voltage detection (when in interrupt mode, and interrupt & reset mode).

R_LVD_Create

Performs initialization necessary to control voltage detector functions.

[Classification]

r_lvd.c

[Syntax]

```
void    R_LVD_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Create_UserInit

Performs user-defined initialization relating to the voltage detector.

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

[Classification]

r_lvd_user.c

[Syntax]

```
void    R_LVD_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Interrupt

Performs processing in response to the voltage detection interrupt INTLVI.

Remark This API function is called as the interrupt process corresponding to the voltage detection interrupt INTLVI.

[Classification]

r_lvd_user.c

[Syntax]

```
void    R_LVD_Interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_InterruptMode_Start

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

[Classification]

r_lvd.c

[Syntax]

```
void    R_LVD_InterruptMode_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

APPENDIX D INDEX

A

A/D Converter ... 145
- R_ADC_Create ... 146
- R_ADC_Create_UserInit ... 147
- R_ADC_Get_Result ... 158
- R_ADC_Get_Result_8bit ... 159
- R_ADC_Interrupt ... 148
- R_ADC_Set_ADChannel ... 154
- R_ADC_Set_ComparatorOff ... 150
- R_ADC_Set_ComparatorOn ... 149
- R_ADC_Set_PowerOff ... 153
- R_ADC_Set_SnoozeOff ... 156
- R_ADC_Set_SnoozeOn ... 155
- R_ADC_Set_TestChannel ... 157
- R_ADC_Start ... 151
- R_ADC_Stop ... 152

[All Output Messages] tab ... 49

API functions ... 59
- A/D Converter ... 145
- Clock Generator ... 69
- Clock Output/Buzzer Output ... 242
- Comparator ... 236
- D/A Converter ... 160
- Data Transfer Controller ... 247
- DMA Controller ... 257
- Event Link Controller ... 253
- Interrupt ... 78
- Interval Timer ... 229
- Port ... 75
- Real-time Clock ... 208
- Serial ... 89
- Timer ... 167
- Voltage Detector ... 264
- Watchdog Timer ... 203

B

Browse For Folder dialog box ... 51

C

Clock Generator ... 69
- R_CGC_Create ... 70
- R_CGC_Create_UserInit ... 71
- R_CGC_Get_ResetSource ... 72
- R_CGC_Set_ClockMode ... 73
- R_CGC_Set_CRCOn ... 74

Clock Output/Buzzer Output ... 242
- R_PCLBUZn_Create ... 243
- R_PCLBUZn_Create_UserInit ... 244
- R_PCLBUZn_Start ... 245
- R_PCLBUZn_Stop ... 246

Code Generator panel ... 41

Code Generator Preview panel ... 44

[Code Generator] tab ... 50

Comparator ... 236
- R_COMP_Create ... 237
- R_COMP_Create_UserInit ... 238
- R_COMPn_Interrupt ... 239
- R_COMPn_Start ... 240
- R_COMPn_Stop ... 241

D

D/A Converter ... 160
- R_DAC_Create ... 161
- R_DAC_Create_UserInit ... 162
- R_DACn_Set_ConversionValue ... 166
- R_DACn_Start ... 163
- R_DACn_Stop ... 164
- R_DAC_Set_PowerOff ... 165

Data Transfer Controller ... 247
- R_DTC_Create ... 248
- R_DTC_Create_UserInit ... 249
- R_DTCn_Start ... 250
- R_DTCn_Stop ... 251
- R_DTC_Set_PowerOff ... 252

DMA Controller ... 257
- R_DMACn_Create ... 258

R_DMACn_Create_UserInit ... 259
R_DMACn_Interrupt ... 260
R_DMACn_SoftwareTriggerOn ... 263
R_DMACn_Start ... 261
R_DMACn_Stop ... 262

E

Event Link Controller ... 253
R_ELC_Create ... 254
R_ELC_Create_UserInit ... 255
R_ELC_Stop ... 256

F

[File Setting] tab ... 40
Functions ... 11
Code Generator ... 11

G

[Generation] tab ... 37

I

Interrupt ... 78
R_INTC_Create ... 79
R_INTC_Create_UserInit ... 80
R_INTCn_Interrupt ... 81
R_INTCn_Start ... 82
R_INTCn_Stop ... 83
R_KEY_Create ... 84
R_KEY_Create_UserInit ... 85
R_KEY_Interrupt ... 86
R_KEY_Start ... 87
R_KEY_Stop ... 88
Interval Timer ... 229
R_IT_Create ... 230
R_IT_Create_UserInit ... 231
R_IT_Interrupt ... 232
R_IT_Set_PowerOff ... 235
R_IT_Start ... 233
R_IT_Stop ... 234

M

[Macro Setting] tab ... 39
Main window ... 28

O

Output panel ... 47
[All Output Messages] tab ... 49
[Code Generator] tab ... 50

P

Port ... 75
R_PORT_Create ... 76
R_PORT_Create_UserInit ... 77
Project Tree panel ... 31
Property panel ... 34
[File Setting] tab ... 40
[Generation] tab ... 37
[Macro Setting] tab ... 39

R

R_ADC_Create ... 146
R_ADC_Create_UserInit ... 147
R_ADC_Get_Result ... 158
R_ADC_Get_Result_8bit ... 159
R_ADC_Interrupt ... 148
R_ADC_Set_ADChannel ... 154
R_ADC_Set_ComparatorOff ... 150
R_ADC_Set_ComparatorOn ... 149
R_ADC_Set_PowerOff ... 153
R_ADC_Set_SnoozeOff ... 156
R_ADC_Set_SnoozeOn ... 155
R_ADC_Set_TestChannel ... 157
R_ADC_Start ... 151
R_ADC_Stop ... 152
R_CGC_Create ... 70
R_CGC_Create_UserInit ... 71
R_CGC_Get_ResetSource ... 72
R_CGC_Set_ClockMode ... 73
R_CGC_Set_CRCOn ... 74
R_COMP_Create ... 237
R_COMP_Create_UserInit ... 238
R_COMPn_Interrupt ... 239
R_COMPn_Start ... 240
R_COMPn_Stop ... 241

R_CSImn_Callback_Error ... 117
R_CSImn_Callback_ReceiveEnd ... 116
R_CSImn_Callback_SendEnd ... 115
R_CSImn_Create ... 108
R_CSImn_Interrupt ... 109
R_CSImn_Receive ... 113
R_CSImn_Send ... 112
R_CSImn_Send_Receive ... 114
R_CSImn_Start ... 110
R_CSImn_Stop ... 111
R_DAC_Create ... 161
R_DAC_Create_UserInit ... 162
R_DACn_Set_ConversionValue ... 166
R_DACn_Start ... 163
R_DACn_Stop ... 164
R_DAC_Set_PowerOff ... 165
R_DMACn_Create ... 258
R_DMACn_Create_UserInit ... 259
R_DMACn_Interrupt ... 260
R_DMACn_SoftwareTriggerOn ... 263
R_DMACn_Start ... 261
R_DMACn_Stop ... 262
R_DTC_Create ... 248
R_DTC_Create_UserInit ... 249
R_DTCn_Start ... 250
R_DTCn_Stop ... 251
R_DTC_Set_PowerOff ... 252
Real-time Clock ... 208
R_RTC_Callback_Alarm ... 226
R_RTC_Callback_ConstPeriod ... 220
R_RTC_Create ... 209
R_RTC_Create_UserInit ... 210
R_RTC_Get_AlarmValue ... 225
R_RTC_Get_CounterValue ... 217
R_RTC_Interrupt ... 211
R_RTC_Set_AlarmOff ... 222
R_RTC_Set_AlarmOn ... 221
R_RTC_Set_AlarmValue ... 223
R_RTC_Set_CounterValue ... 216
R_RTC_Set_HourSystem ... 215
R_RTC_Set_PowerOff ... 214
R_RTC_Set_RTC1HZOff ... 228
R_RTC_Set_RTC1HZOn ... 227
R_RTC_Start ... 212
R_RTC_Stop ... 213
R_RTC_Set_ConstPeriodInterruptOff ... 219
R_RTC_Set_ConstPeriodInterruptOn ... 218
R_ELC_Create ... 254
R_ELC_Create_UserInit ... 255
R_ELC_Stop ... 256
R_IICAn_Callback_GetStopCondition ... 144
R_IICAn_Callback_Master_Error ... 138
R_IICAn_Callback_Master_ReceiveEnd ... 137
R_IICAn_Callback_Master_SendEnd ... 136
R_IICAn_Callback_Slave_Error ... 143
R_IICAn_Callback_Slave_ReceiveEnd ... 142
R_IICAn_Callback_Slave_SendEnd ... 141
R_IICAn_Create ... 128
R_IICAn_Create_UserInit ... 129
R_IICAn_Interrupt ... 130
R_IICAn_Master_Receive ... 135
R_IICAn_Master_Send ... 134
R_IICAn_Set_PowerOff ... 133
R_IICAn_Slave_Receive ... 140
R_IICAn_Slave_Send ... 139
R_IICAn_Stop ... 132
R_IICAn_StopCondition ... 131
R_IICmn_Callback_Master_Error ... 127
R_IICmn_Callback_Master_ReceiveEnd ... 126
R_IICmn_Callback_Master_SendEnd ... 125
R_IICmn_Create ... 118
R_IICmn_Interrupt ... 119
R_IICmn_Master_Receive ... 124
R_IICmn_Master_Send ... 123
R_IICmn_StartCondition ... 120
R_IICmn_Stop ... 122
R_IICmn_StopCondition ... 121
R_INTC_Create ... 79
R_INTC_Create_UserInit ... 80
R_INTCn_Interrupt ... 81
R_INTCn_Start ... 82
R_INTCn_Stop ... 83

R_IT_Create ... 230
R_IT_Create_UserInit ... 231
R_IT_Interrupt ... 232
R_IT_Set_PowerOff ... 235
R_IT_Start ... 233
R_IT_Stop ... 234
R_KEY_Create ... 84
R_KEY_Create_UserInit ... 85
R_KEY_Interrupt ... 86
R_KEY_Start ... 87
R_KEY_Stop ... 88
R_LVD_Create ... 265
R_LVD_Create_UserInit ... 266
R_LVD_Interrupt ... 267
R_LVD_InterruptMode_Start ... 268
R_PCLBUZn_Create ... 243
R_PCLBUZn_Create_UserInit ... 244
R_PCLBUZn_Start ... 245
R_PCLBUZn_Stop ... 246
R_PORT_Create ... 76
R_PORT_Create_UserInit ... 77
R_RTC_Callback_Alarm ... 226
R_RTC_Callback_ConstPeriod ... 220
R_RTC_Create ... 209
R_RTC_Create_UserInit ... 210
R_RTC_Get_AlarmValue ... 225
R_RTC_Get_CounterValue ... 217
R_RTC_Interrupt ... 211
R_RTC_Set_AlarmOff ... 222
R_RTC_Set_AlarmOn ... 221
R_RTC_Set_AlarmValue ... 223
R_RTC_Set_CounterValue ... 216
R_RTC_Set_HourSystem ... 215
R_RTC_Set_PowerOff ... 214
R_RTC_Set_RTC1HZOff ... 228
R_RTC_Set_RTC1HZOn ... 227
R_RTC_Start ... 212
R_RTC_Stop ... 213
R_RTC_Set_ConstPeriodInterruptOff ... 219
R_RTC_Set_ConstPeriodInterruptOn ... 218
R_SAU0_Set_SnoozeOff ... 95
R_SAU0_Set_SnoozeOn ... 94
R_SAUm_Create ... 91
R_SAUm_Create_UserInit ... 92
R_SAUm_Set_PowerOff ... 93
R_TAUm_Channeln_Get_PulseWidth ... 180
R_TAUm_Channeln_Higher8bits_Interrupt ... 172
R_TAUm_Channeln_Higher8bits_Start ... 174
R_TAUm_Channeln_Higher8bits_Stop ... 177
R_TAUm_Channeln_Interrupt ... 171
R_TAUm_Channeln_Lower8bits_Start ... 175
R_TAUm_Channeln_Lower8bits_Stop ... 178
R_TAUm_Channeln_SoftwareTriggerOn ... 181
R_TAUm_Channeln_Start ... 173
R_TAUm_Channeln_Stop ... 176
R_TAUm_Create ... 169
R_TAUm_Create_UserInit ... 170
R_TAUm_Set_PowerOff ... 179
R_TMR_RDn_Create ... 182
R_TMR_RDn_Create_UserInit ... 183
R_TMR_RDn_Get_PulseWidth ... 188
R_TMR_RDn_Interrupt ... 184
R_TMR_RDn_Set_PowerOff ... 187
R_TMR_RDn_Start ... 185
R_TMR_RDn_Stop ... 186
R_TMR_RGn_Create ... 189
R_TMR_RGn_Create_UserInit ... 190
R_TMR_RGn_Get_PulseWidth ... 195
R_TMR_RGn_Interrupt ... 191
R_TMR_RGn_Set_PowerOff ... 194
R_TMR_RGn_Start ... 192
R_TMR_RGn_Stop ... 193
R_TMR_RJ0_Create ... 196
R_TMR_RJ0_Create_UserInit ... 197
R_TMR_RJ0_Get_PulseWidth ... 202
R_TMR_RJ0_Interrupt ... 198
R_TMR_RJ0_Set_PowerOff ... 201
R_TMR_RJ0_Start ... 199
R_TMR_RJ0_Stop ... 200
R_UARTn_Callback_Error ... 106
R_UARTn_Callback_ReceiveEnd ... 105
R_UARTn_Callback_SendEnd ... 104

R_UARTn_Callback_SoftwareOverRun ... 107
R_UARTn_Create ... 96
R_UARTn_Interrupt_Error ... 99
R_UARTn_Interrupt_Receive ... 98
R_UARTn_Interrupt_Send ... 97
R_UARTn_Receive ... 103
R_UARTn_Send ... 102
R_UARTn_Start ... 100
R_UARTn_Stop ... 101
R_WDT_Create ... 204
R_WDT_Create_UserInit ... 205
R_WDT_Interrupt ... 206
R_WDT_Restart ... 207

S

Save As dialog box ... 52
Serial ... 89
R_CSImn_Callback_Error ... 117
R_CSImn_Callback_ReceiveEnd ... 116
R_CSImn_Create ... 108
R_CSImn_Interrupt ... 109
R_CSImn_Receive ... 113
R_CSImn_Send ... 112
R_CSImn_Callback_SendEnd ... 115
R_CSImn_Send_Receive ... 114
R_CSImn_Start ... 110
R_CSImn_Stop ... 111
R_IICAn_Callback_GetStopCondition ... 144
R_IICAn_Callback_Master_Error ... 138
R_IICAn_Callback_Master_ReceiveEnd ... 137
R_IICAn_Callback_Master_SendEnd ... 136
R_IICAn_Callback_Slave_Error ... 143
R_IICAn_Callback_Slave_ReceiveEnd ... 142
R_IICAn_Callback_Slave_SendEnd ... 141
R_IICAn_Create ... 128
R_IICAn_Create_UserInit ... 129
R_IICAn_Interrupt ... 130
R_IICAn_Master_Receive ... 135
R_IICAn_Master_Send ... 134
R_IICAn_Set_PowerOff ... 133
R_IICAn_Slave_Receive ... 140
R_IICAn_Slave_Send ... 139
R_IICAn_Stop ... 132
R_IICAn_StopCondition ... 131
R_IICmn_Callback_Master_Error ... 127
R_IICmn_Callback_Master_ReceiveEnd ... 126
R_IICmn_Callback_Master_SendEnd ... 125
R_IICmn_Create ... 118
R_IICmn_Interrupt ... 119
R_IICmn_Master_Receive ... 124
R_IICmn_Master_Send ... 123
R_IICmn_StartCondition ... 120
R_IICmn_Stop ... 122
R_IICmn_StopCondition ... 121
R_SAU0_Set_SnoozeOff ... 95
R_SAU0_Set_SnoozeOn ... 94
R_SAUm_Create ... 91
R_SAUm_Create_UserInit ... 92
R_SAUm_Set_PowerOff ... 93
R_UARTn_Callback_Error ... 106
R_UARTn_Callback_ReceiveEnd ... 105
R_UARTn_Callback_SendEnd ... 104
R_UARTn_Callback_SoftwareOverRun ... 107
R_UARTn_Create ... 96
R_UARTn_Interrupt_Error ... 99
R_UARTn_Interrupt_Receive ... 98
R_UARTn_Interrupt_Send ... 97
R_UARTn_Receive ... 103
R_UARTn_Send ... 102
R_UARTn_Start ... 100
R_UARTn_Stop ... 101

T

Timer ... 167
R_TAUm_Channeln_Get_PulseWidth ... 180
R_TAUm_Channeln_Higher8bits_Interrupt ... 172
R_TAUm_Channeln_Higher8bits_Stop ... 177
R_TAUm_Channeln_Interrupt ... 171
R_TAUm_Channeln_Lower8bits_Start ... 175
R_TAUm_Channeln_Lower8bits_Stop ... 178
R_TAUm_Channeln_SoftwareTriggerOn ... 181
R_TAUm_Channeln_Start ... 173

R_TAUm_ChannelN_Stop ... 176
R_TAUm_Create ... 169
R_TAUm_Create_UserInit ... 170
R_TAUm_Set_PowerOff ... 179
R_TMR_RDn_Create ... 182
R_TMR_RDn_Create_UserInit ... 183
R_TMR_RDn_Get_PulseWidth ... 188
R_TMR_RDn_Interrupt ... 184
R_TMR_RDn_Set_PowerOff ... 187
R_TMR_RDn_Start ... 185
R_TMR_RDn_Stop ... 186
R_TMR_RGn_Create ... 189
R_TMR_RGn_Create_UserInit ... 190
R_TMR_RGn_Get_PulseWidth ... 195
R_TMR_RGn_Interrupt ... 191
R_TMR_RGn_Set_PowerOff ... 194
R_TMR_RGn_Start ... 192
R_TMR_RGn_Stop ... 193
R_TMR_RJ0_Create ... 196
R_TMR_RJ0_Create_UserInit ... 197
R_TMR_RJ0_Get_PulseWidth ... 202
R_TMR_RJ0_Interrupt ... 198
R_TMR_RJ0_Set_PowerOff ... 201
R_TMR_RJ0_Start ... 199
R_TMR_RJ0_Stop ... 200
R_TAUm_ChannelN_Higher8bits_Start ... 174

V

Voltage Detector ... 264
R_LVD_Create ... 265
R_LVD_InterruptMode_Start ... 268
R_LVD_Create_UserInit ... 266
R_LVD_Interrupt ... 267

W

Watchdog Timer ... 203
R_WDT_Create ... 204
R_WDT_Create_UserInit ... 205
R_WDT_Interrupt ... 206
R_WDT_Restart ... 207
Window reference ... 27

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jun 01, 2011	-	First Edition issued

CubeSuite+ V1.00.01
User’s Manual: RL78 Design

Publication Date: Rev.1.00 Jun 01, 2011

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: +44-1628-585-100, Fax: +44-1628-585-900

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.
7F, No. 363 Fu Shing North Road Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

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 1.0

CubeSuite+ V1.00.01

Renesas Electronics Corporation

R20UT0548EJ0100