User's Manual

RENESAS

CubeSuite+ V1.03.00

Integrated Development Environment

User's Manual: 78K0R Design

Target Device

78K0R Microcontrollers

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

Renesas Electronics
www.renesas.com

Rev.1.00 Sep 2012

How to Use This Manual

This manual describes the role of the CubeSuite+ integrated development environment for developing applications and systems for 78K0R microcontrollers, and provides an outline of its features.

CubeSuite+ is an integrated development environment (IDE) for 78K0R microcontrollers, integrating the necessary tools for the development phase of software (e.g. design, implementation, and debugging) into a single platform.

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

Readers

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

Purpose

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

Organization

This manual can be broadly divided into the following units.

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

How to Read This Manual

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

Conventions

Data significance:	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	R20UT2133E
	V850 Design	R20UT2134E
	R8C Design	R20UT2135E
	RL78 Design	R20UT2136E
	78K0R Design	This manual
	78K0 Design	R20UT2138E
	RX Coding	R20UT0767E
	V850 Coding	R20UT0553E
	Coding for CX Compiler	R20UT2139E
	R8C Coding	R20UT0576E
	RL78, 78K0R Coding	R20UT2140E
	78K0 Coding	R20UT2141E
	RX Build	R20UT0768E
	V850 Build	R20UT0557E
	Build for CX Compiler	R20UT2142E
	R8C Build	R20UT0575E
	RL78, 78K0R Build	R20UT2143E
	78K0 Build	R20UT0783E
	RX Debug	R20UT2175E
	V850 Debug	R20UT2144E
	R8C Debug	R20UT0770E
	RL78 Debug	R20UT2145E
	78K0R Debug	R20UT0732E
	78K0 Debug	R20UT0731E
	Analysis	R20UT2146E
	Message	R20UT2147E

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

TABLE OF CONTENTS

CHAPTER 1 GENERAL

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

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

1.1 Overview

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

1.2 Features

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

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

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

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

CHAPTER 2 FUNCTIONS (Pin Configurator)

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

2.1 Overview

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

The following sections describe the operation procedures for Pin Configurator.

(1) Start CubeSuite+

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

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

(2) Create/Open project

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

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

(3) Open Device Pin List Panel

Open the Device Pin List panel, where you enter information on the pins of the microcontroller.

(a) Select item

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

(b) Change display order

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

(c) Add column

Allows you to add columns to the device pin list.

(d) Delete column

Allows you to delete columns from the device pin list.

(4) Open Device Top View Panel

Open the Device Top View panel, where you can confirm the information entered for the pins.

(a) Select shape of microcontroller

Allows you to select the shape of the microcontroller displayed in the Device Top View panel.

(b) Select color

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

(c) Select popup information

Allows you to select the type of information that popups when you move the mouse cursor over each pin in the Device Top View panel.

(d) Select additional information

Select the type of information to display in Pin area of the Device Top View panel.

(5) Enter Information

Enter information on the pins of the microcontroller in the Device Pin List panel.

(6) Output Report Files

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

(a) Output device pin list

Output a device pin list.

(b) Output device top view

Output a device top view.

(7) Save project

Save a project.

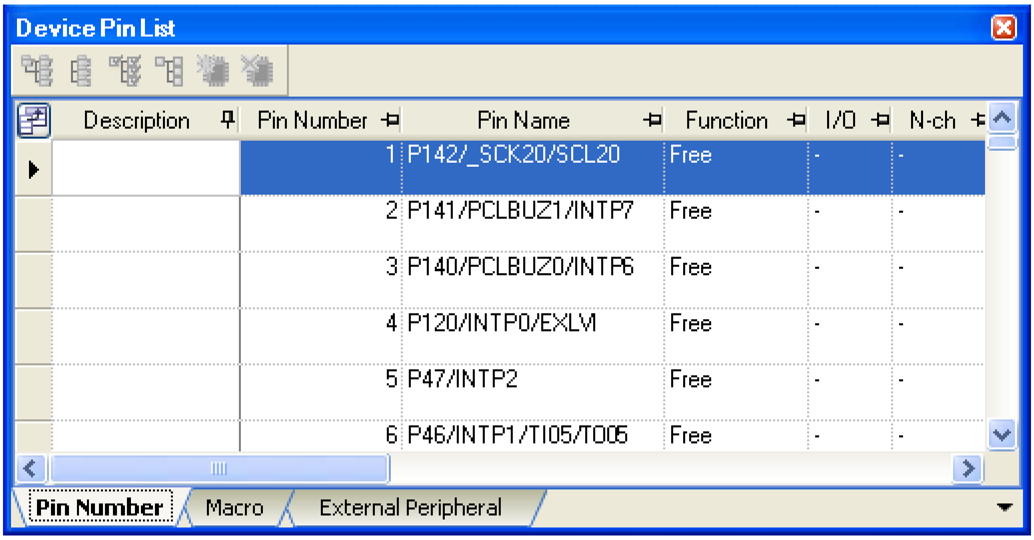
Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Save project".

2.2 Open Device Pin List Panel

Open the Device Pin List panel, where you enter information on the pins of the microcontroller.

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

Figure 2-1. Open Device Pin List Panel



Remarks 1. If an unsupported microcontroller is defined in the project for Pin Configurator, then "[Pin Configurator (Design Tool)] node" will hide under [*Project name* (Project)] in the Project Tree panel.

2. The Device Pin List panel consists of three tabs. Selecting one of the tabs changes the order in which "information on each pin of the microcontroller" is displayed.

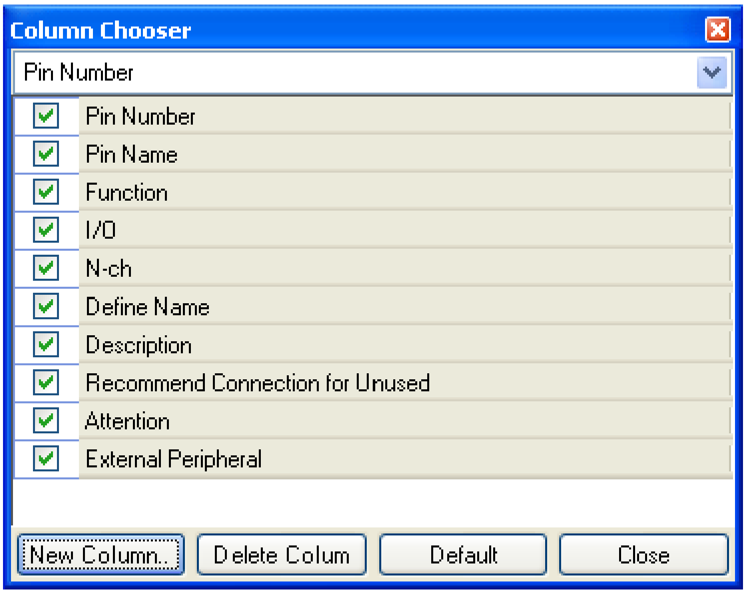
- [Pin Number] tab
  Information on each pin of the microcontroller is displayed in the order of pin number.
- [Macro] tab
  Information on each pin of the microcontroller is displayed in the order it was grouped into peripheral functions.
- [External Peripheral] tab
  Information about the pins connected to external peripherals is displayed in order grouped at the external-peripheral component level.

2.2.1 Select item

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

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

Figure 2-2. Select Item



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

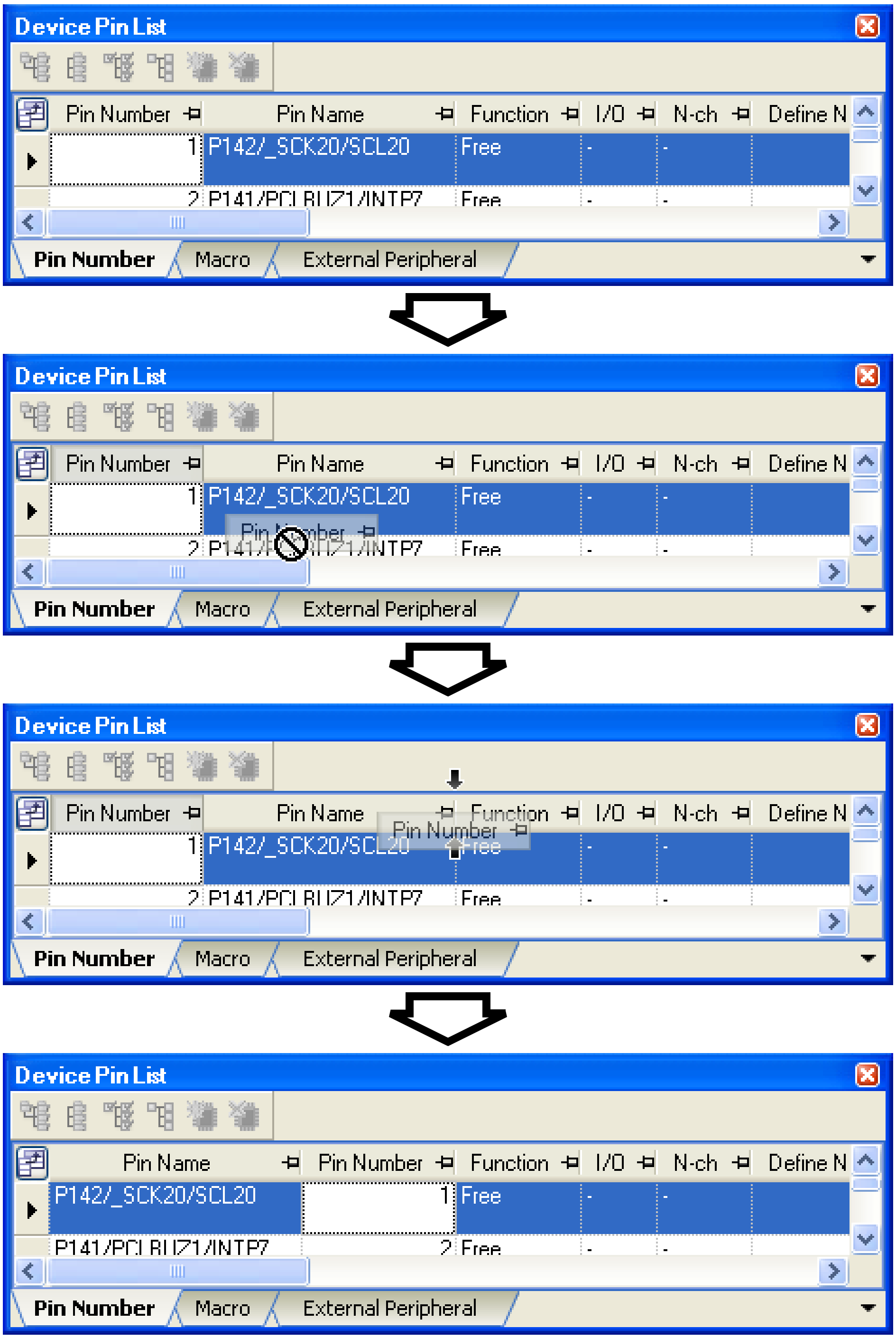
Table 2-1. Select Item

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

2.2.2 Change display order

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

Figure 2-3. Change Display Order



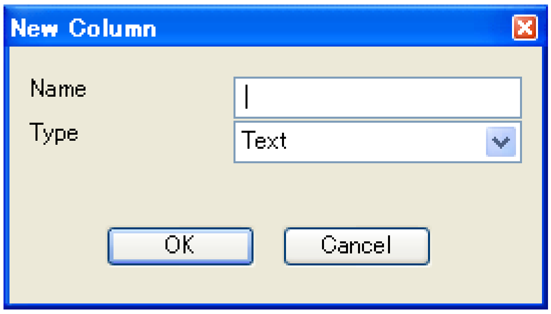
Remark To change the display order, click the button in the upper left of the device pin list. The Column Chooser dialog box opens. Drag an item displayed in the dialog's select Items to display area, and drop it to the desired destination in the device pin list. This will change the display order.

2.2.3 Add column

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

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

Figure 2-4. Add Column



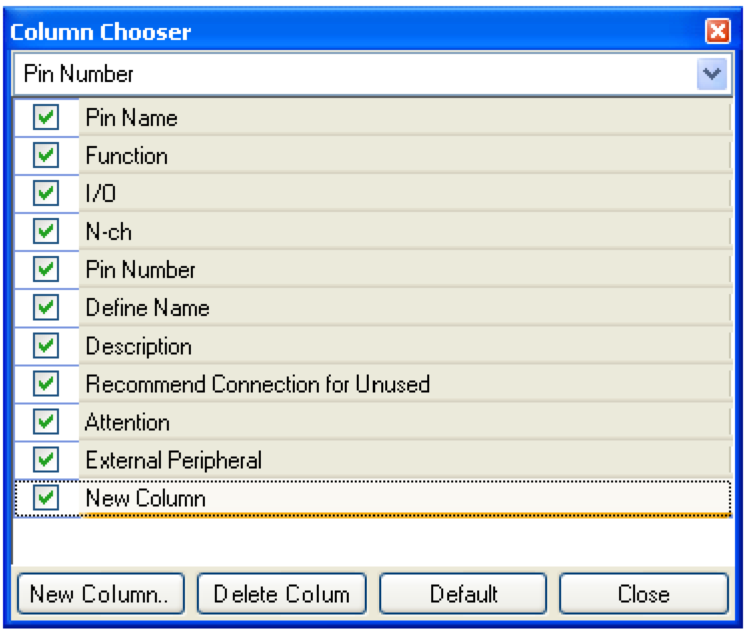
Remark On the device pin list, adding columns to the first level of [Macro] tab, [External Peripheral] tab is restricted.

2.2.4 Delete column

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

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

Figure 2-5. Delete Column



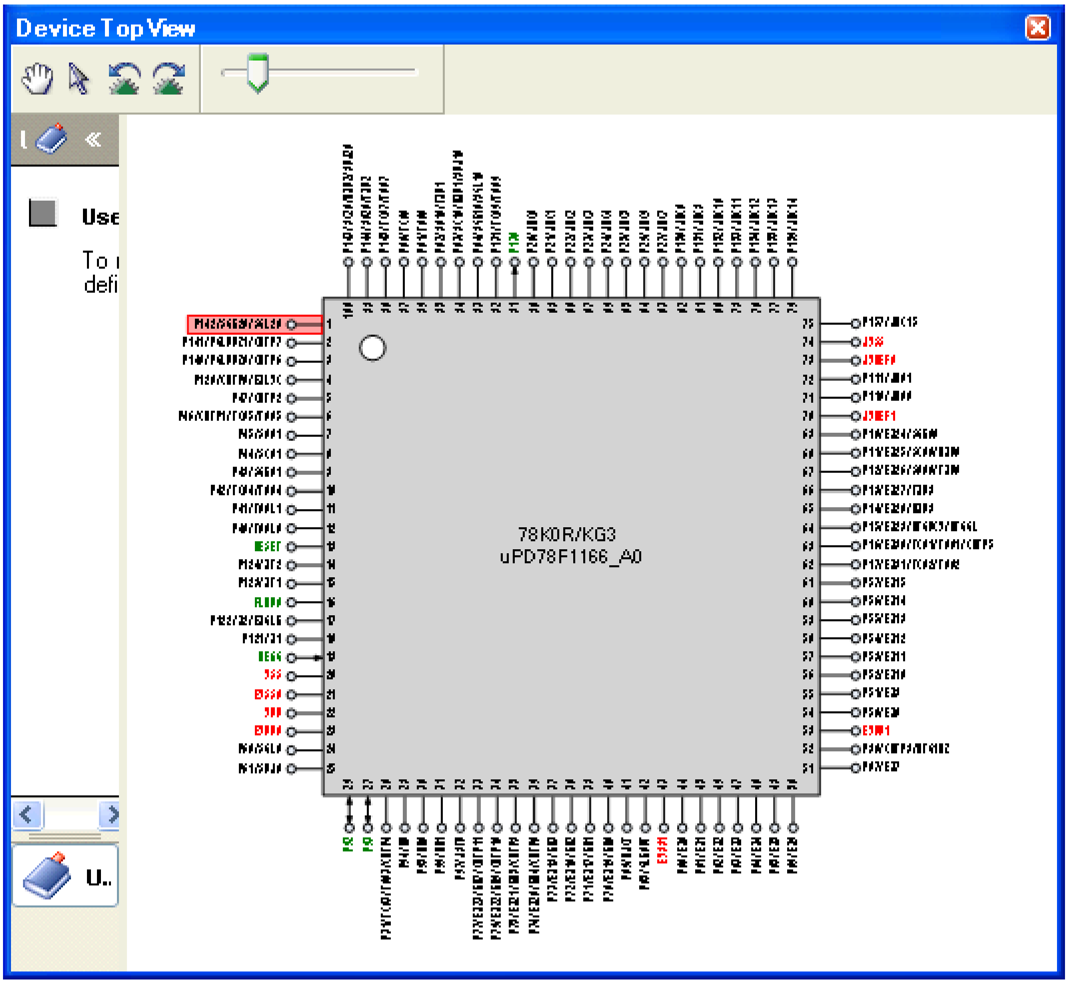
Remark You can only delete the column which you added using the New Column dialog box.

2.3 Open Device Top View Panel

Open the Device Top View panel, where you can confirm the information entered for the pins of the microcontroller.

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

Figure 2-6. Open Device Top View Panel



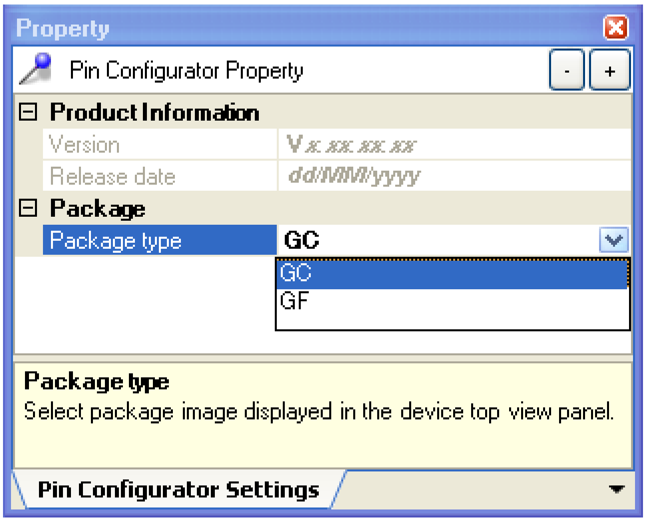
Remark In the Property panel, on the [Pin Configurator Settings] tab, if "BGA" is selected for the Package type, then Device Top View panel cannot be opened.

2.3.1 Select shape of microcontroller

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

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

Figure 2-7. Select Shape of Microcontroller



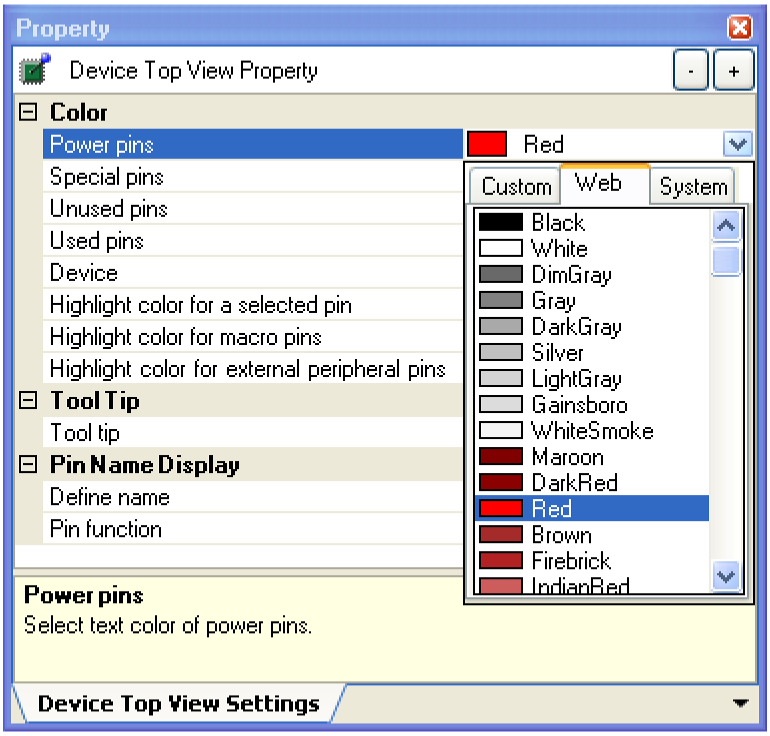
Remark Selection of the shape of the microcontroller is made using the order name (such as GC and GF).

2.3.2 Select color

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

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

Figure 2-8. Select Color



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

Table 2-2. Select Color

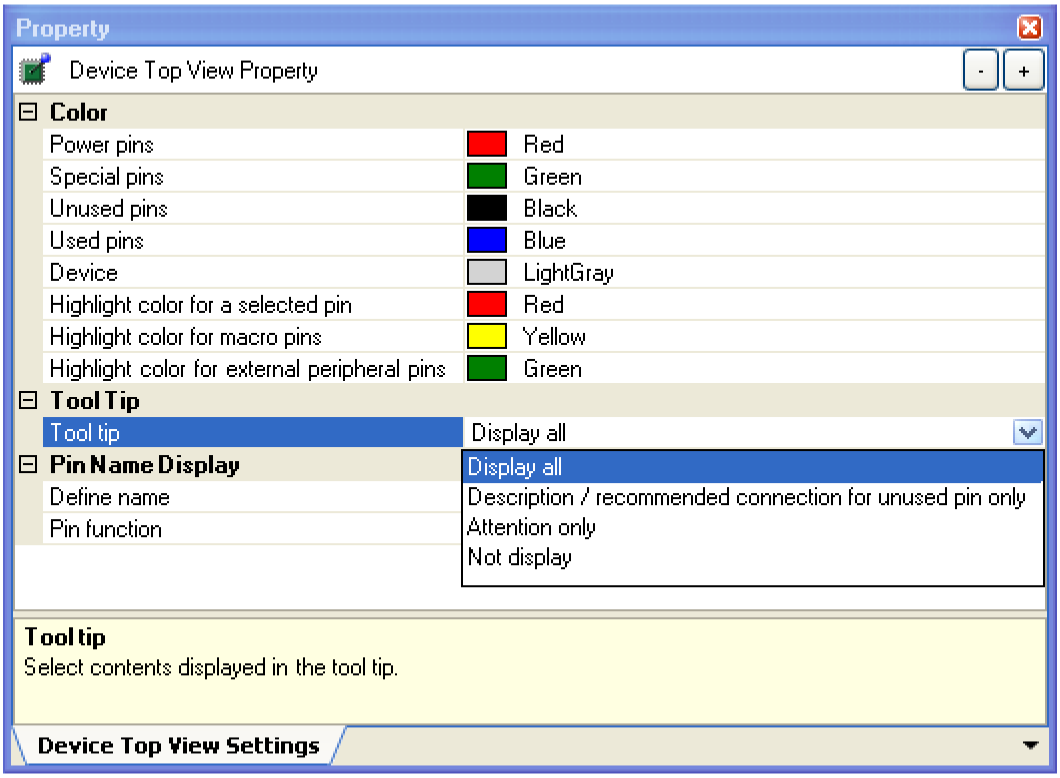
Item	Outline
Power pins	Selects the display color for power pins (pins whose use is limited to power).
Special pins	Selects the display color for special pins (pins with specified uses).
Unused pins	Selects the display color for unused pins (dual-use pins with no use set in the Device Pin List panel).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

2.3.3 Select popup information

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

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

Figure 2-9. Select Popup Information



Remark Popup information is selected from the following four types.

Table 2-3. Select Popup Information

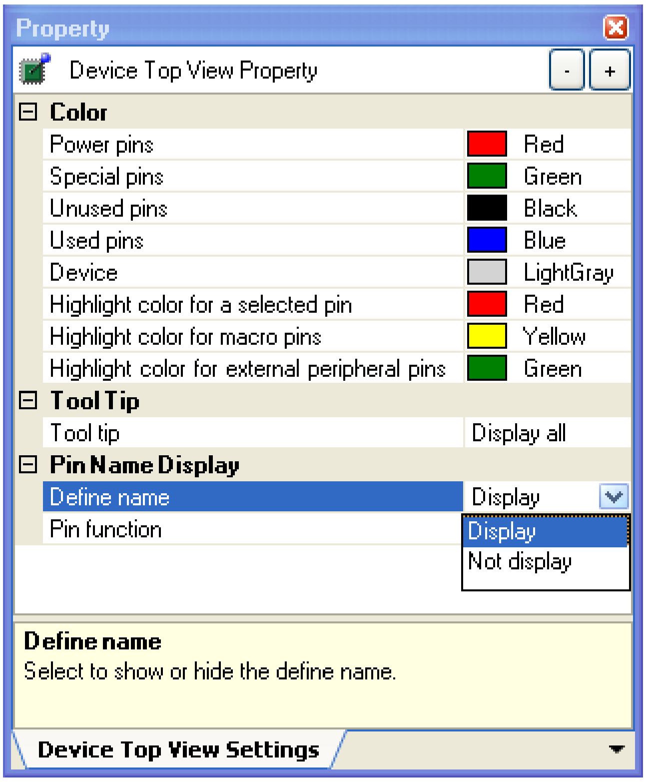
Popup Information	Outline
Display all	Displays the "Description", "Recommend Connection for Unused", and "Attention" strings for the device pin list.
Description / recommended connection for unused pin only	Displays the "Description", and "Recommend Connection for Unused" strings for the device pin list.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

2.3.4 Select additional information

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

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

Figure 2-10. Select Additional Information



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

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

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

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

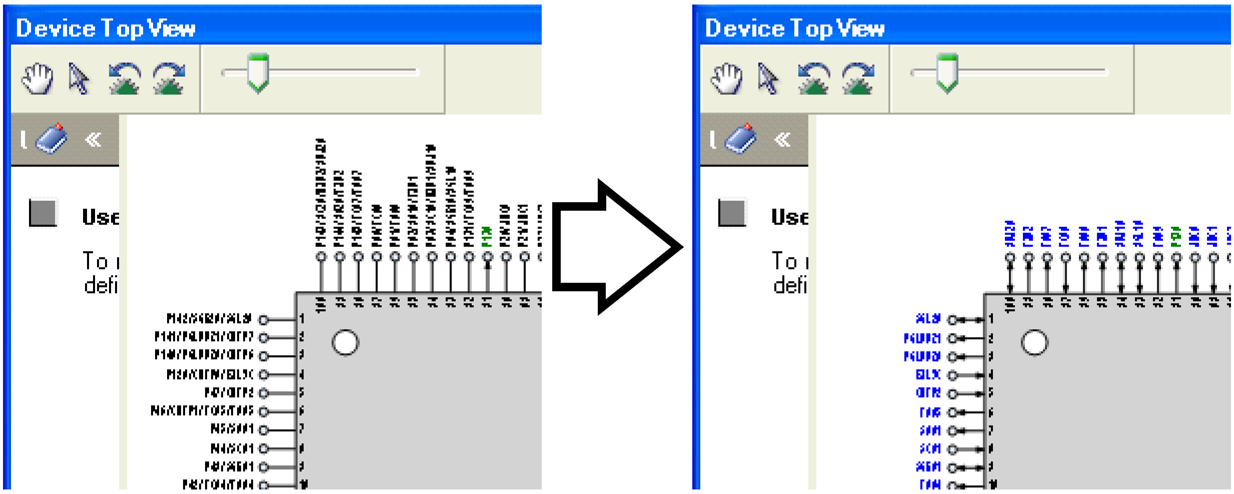
2.4 Enter Information

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

Remarks 1. You cannot add information in the "Pin Number" column, "Pin Name" column, "Description" column, "Recommend Connection for Unused" column and "Attention" column because they contain fixed information.

2. If the "Free" in the "Function" column is changed to a specific pin name, color of the corresponding pin in the Device Top View panel changes from the "color representing the unused pins" to the "color representing the used pins" selected by clicking [Device Top View Settings] tab >> [Color] in the Property panel.

Figure 2-11. Change in Displayed Color



2.5 Output Report Files

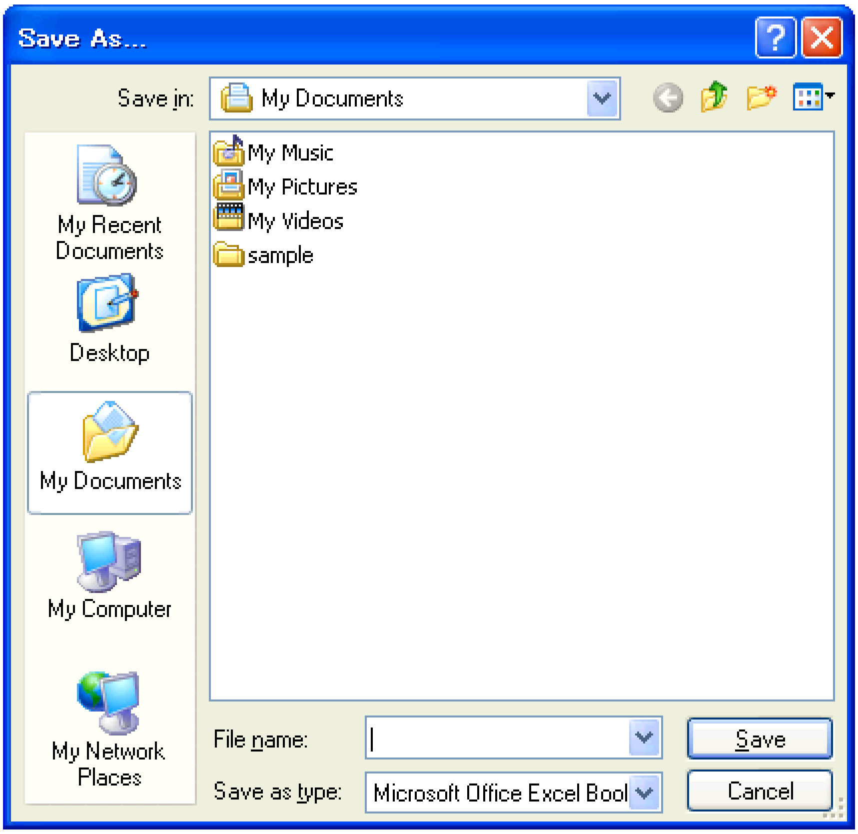
Output report files (files containing information configured using Pin Configurator: device pin list and device top view) to the specified folder.

2.5.1 Output device pin list

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

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

Figure 2-12. Output Device Pin List



Remarks 1. If a device pin list has been already output, that list will be overwritten by selecting [File] menu >> [Save Pin List].

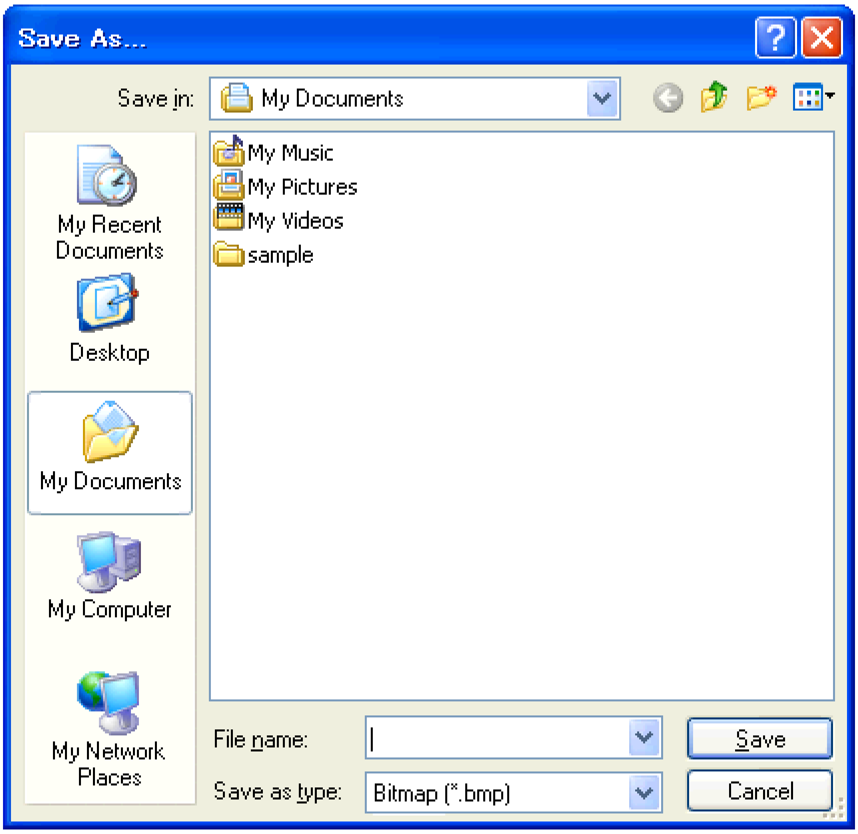
2. The output format for the device pin list is limited to Microsoft Office Excel Book.

2.5.2 Output device top view

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

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

Figure 2-13. Output Device Top View



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

CHAPTER 3 FUNCTIONS (Code Generator)

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

3.1 Overview

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

The following sections describe the operation procedures for Code Generator.

(1) Start CubeSuite+

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

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

(2) Create/Open project

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

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

(3) Open Code Generator Panel

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

(4) Enter Information

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

(5) Confirm Source Code

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

(6) Output Source Code

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

(7) Output Report Files

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

(8) Save project

Save a project.

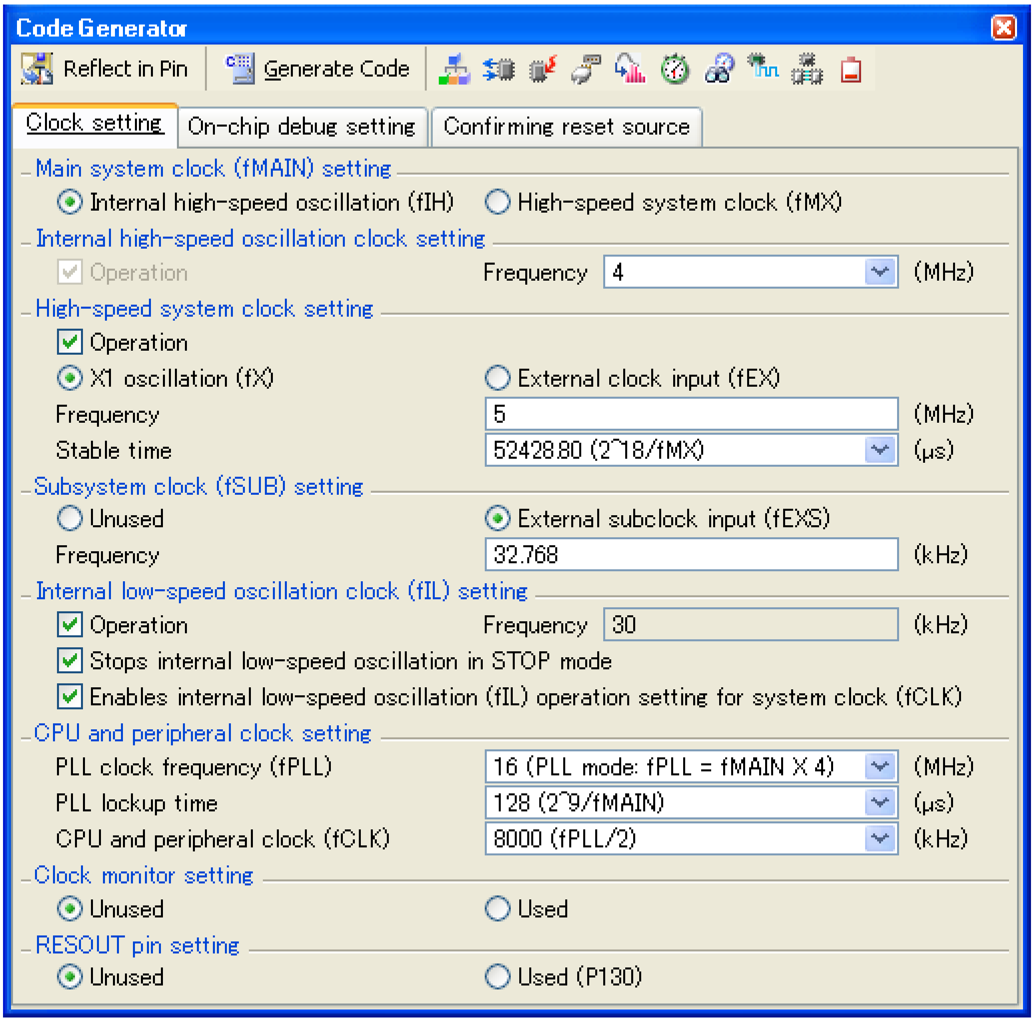
Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Save project".

3.2 Open Code Generator Panel

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

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

Figure 3-1. Open Code Generator Panel



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

3.3 Enter Information

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

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

3.3.1 Input rule

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

(1) Character set

Character sets that are allowed to input are as follows.

Table 3-1. List of Character Set

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

(2) Number

Notations allowed when entering numbers are as follows.

Table 3-2. List of Notation

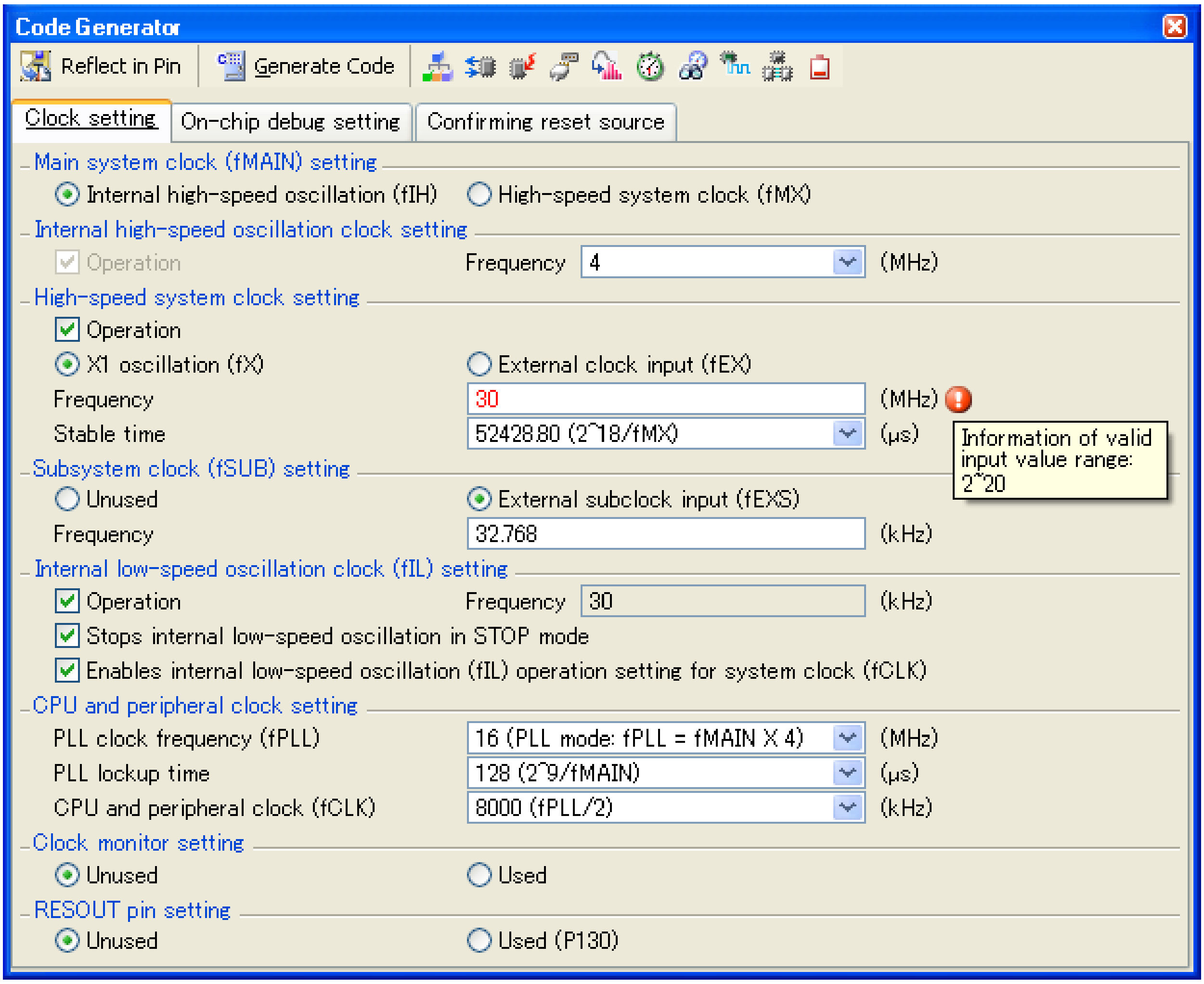
Notation	Outline
Decimal number	A numeric value that starts with a number between 1 and 9 and followed by numbers between 0 and 9, and the numeric value 0
Hex number	A numeric value that starts with 0x and followed by a combination of numbers from 0 to 9 and characters from A to F (characters are not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

3.3.2 Icon indicating incorrect entry

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

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

Figure 3-2. Icon Indicating Incorrect Entry

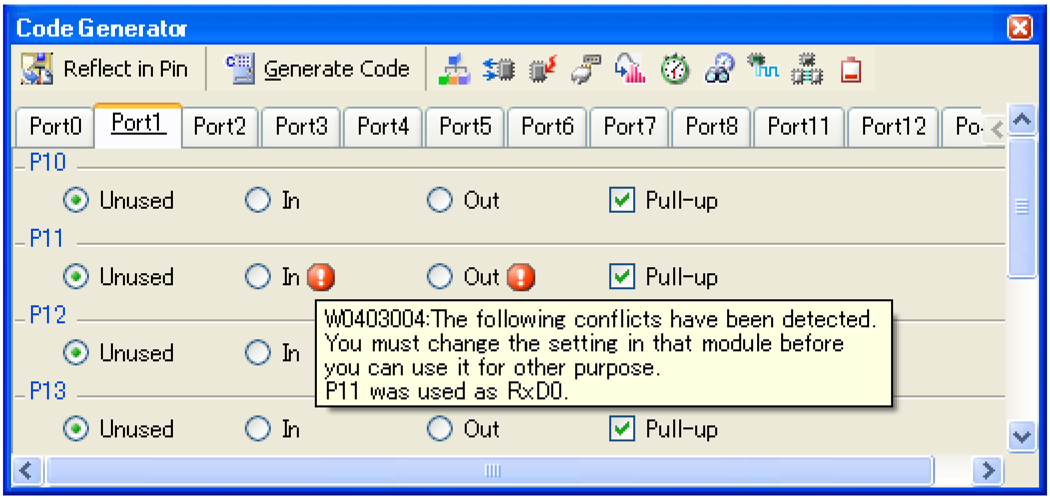


3.3.3 Icon indicating pin conflict

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

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

Figure 3-3. Icon Indicating Pin Conflict

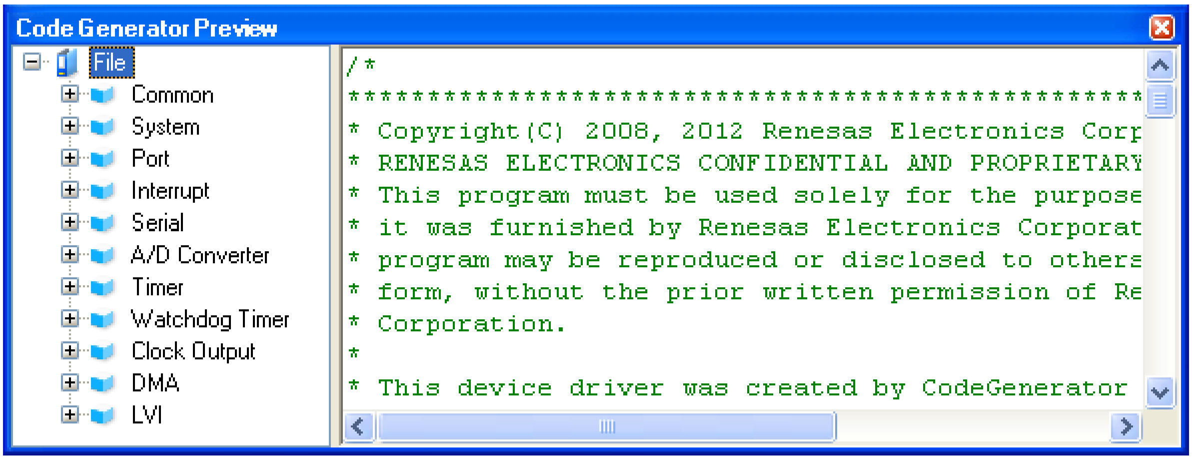


3.4 Confirm Source Code

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

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

Figure 3-4. Confirm Source Code



Remarks 1. You can change the source code to be displayed by selecting the source file name or API function name in the Code Generator Preview panel.

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

Table 3-3. Color of Source Code

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

3. You cannot edit the source code within the Code Generator Preview panel.

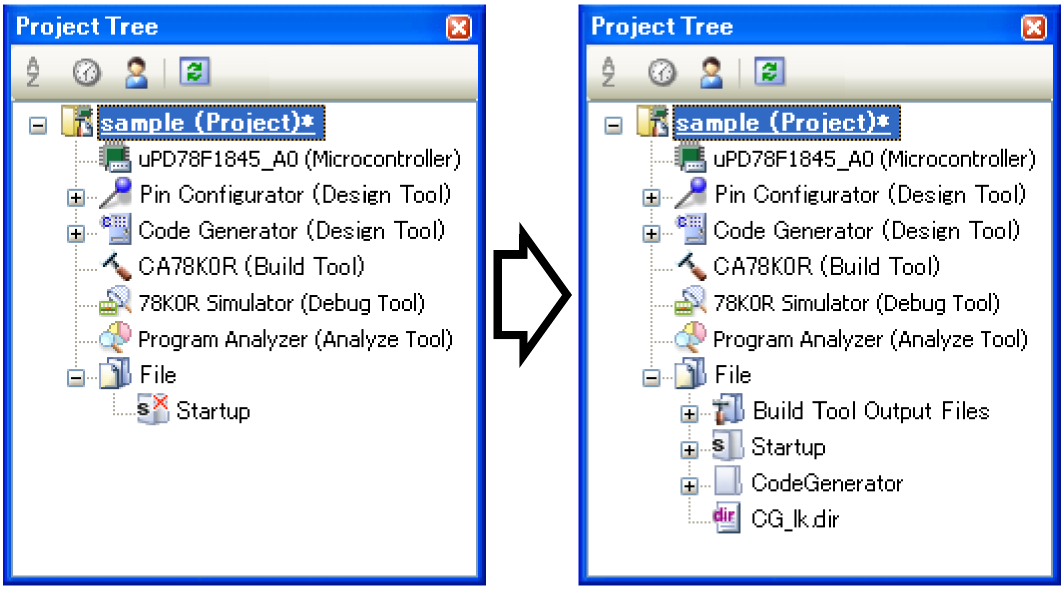
4. For some of the API functions (such as API functions for serial array units), values such as the SFR register value are calculated and finalized when the source code is generated (when the Generate Code button on the Code Generator panel is pressed). For this reason, the source code displayed in the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

3.5 Output Source Code

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

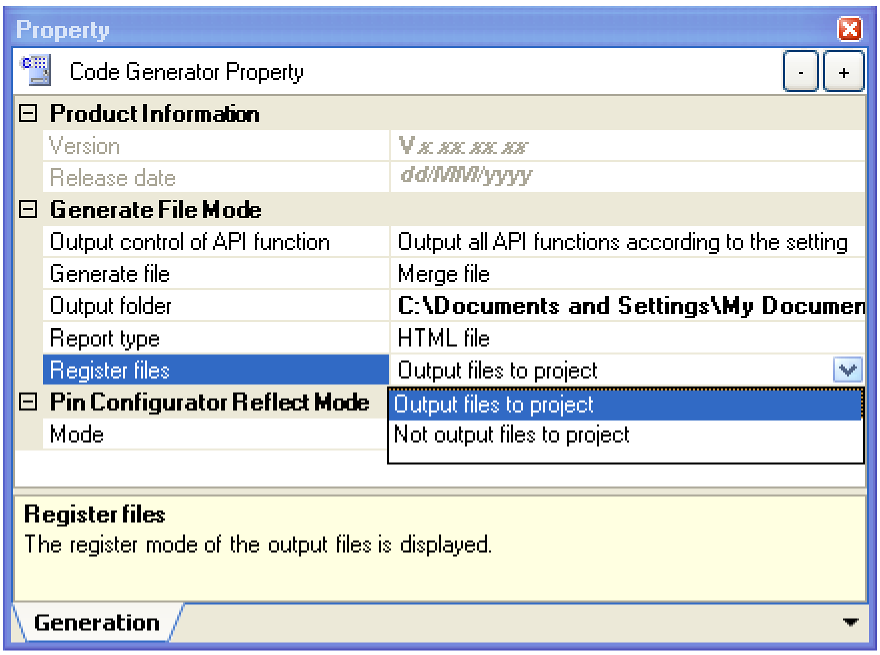
The destination folder for the source code is specified by clicking [Generation] tab >> [Output folder] in the Property panel.

Figure 3-5. Output Source Code



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

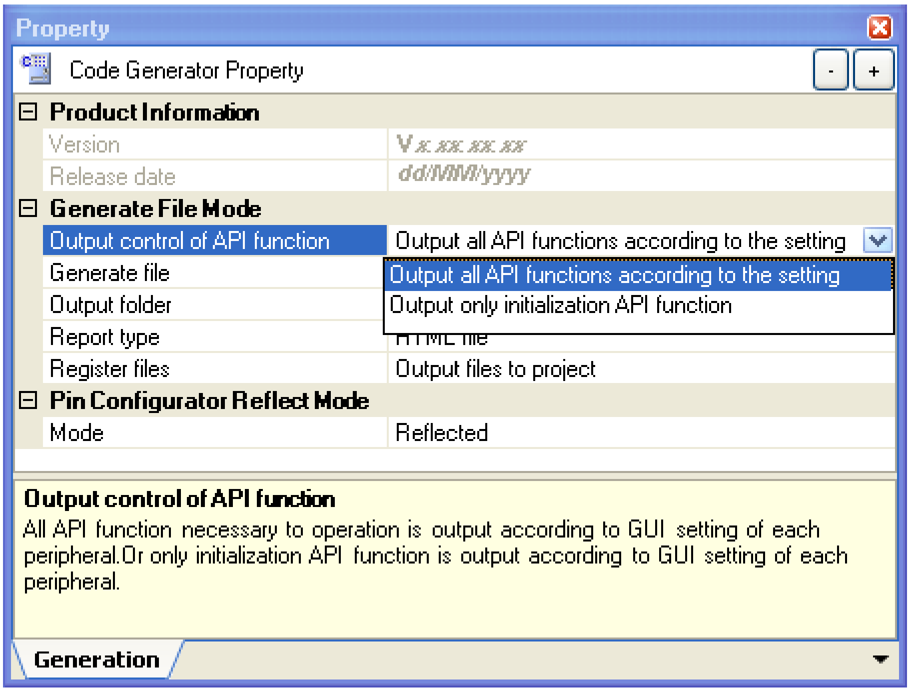
Figure 3-6. Configure Whether to Register



3.5.1 Setting that determines whether or not to generate source code

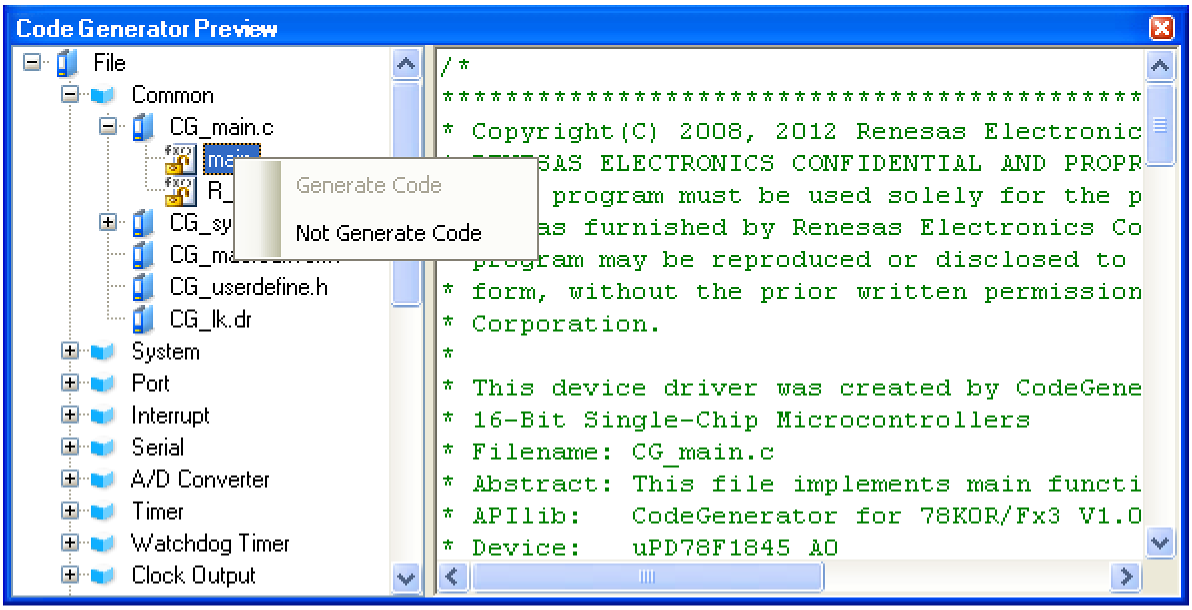
You can set the type of output API functions (all API functions or only initialization API functions) by selecting [Output all API function according to the setting/Output only initialization API function] from [Generation] tab >> [Output control of API function] in the Property panel.

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



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

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



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

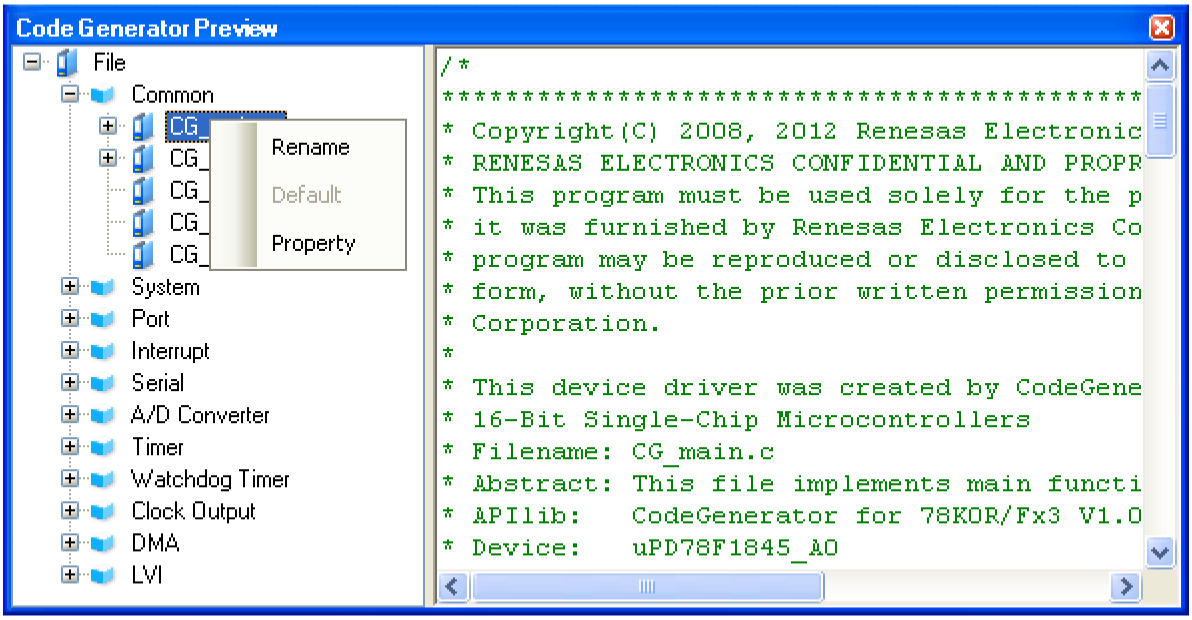
Table 3-4. Setting That Determines Whether or Not to Generate Source Code

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

3.5.2 Change file name

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

Figure 3-9. Change File Name

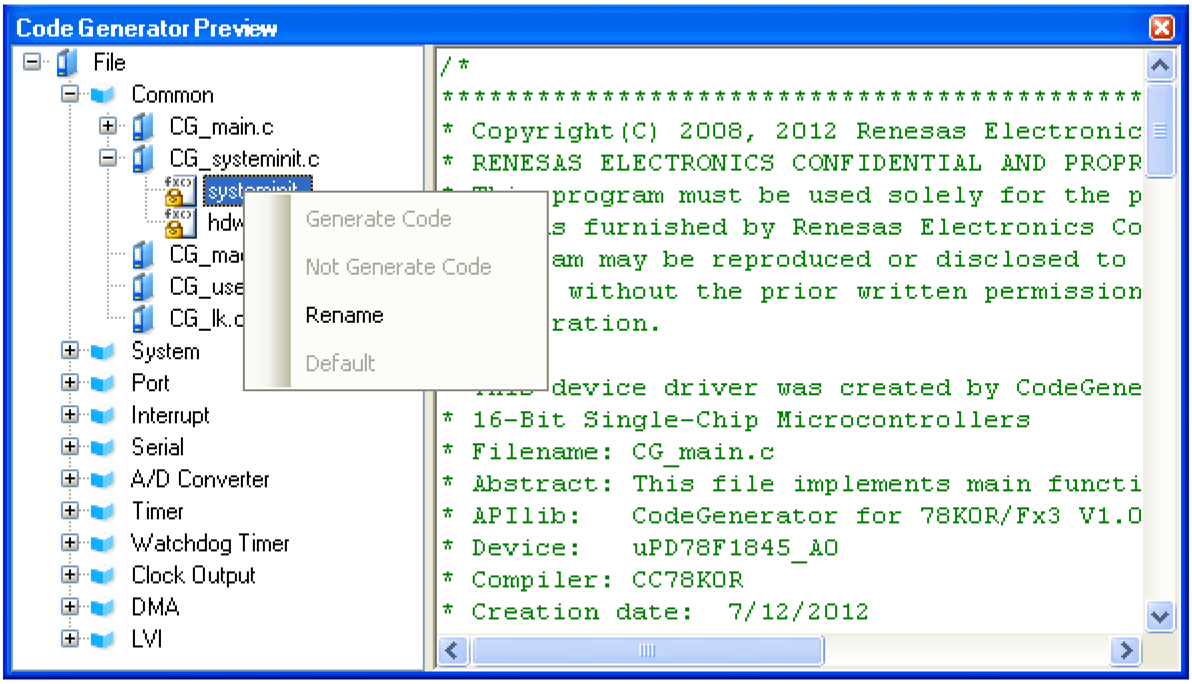


Remark To restore the default file name defined by Code Generator, select [Default] from the context menu.

3.5.3 Change API function name

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

Figure 3-10. Change API Function Name

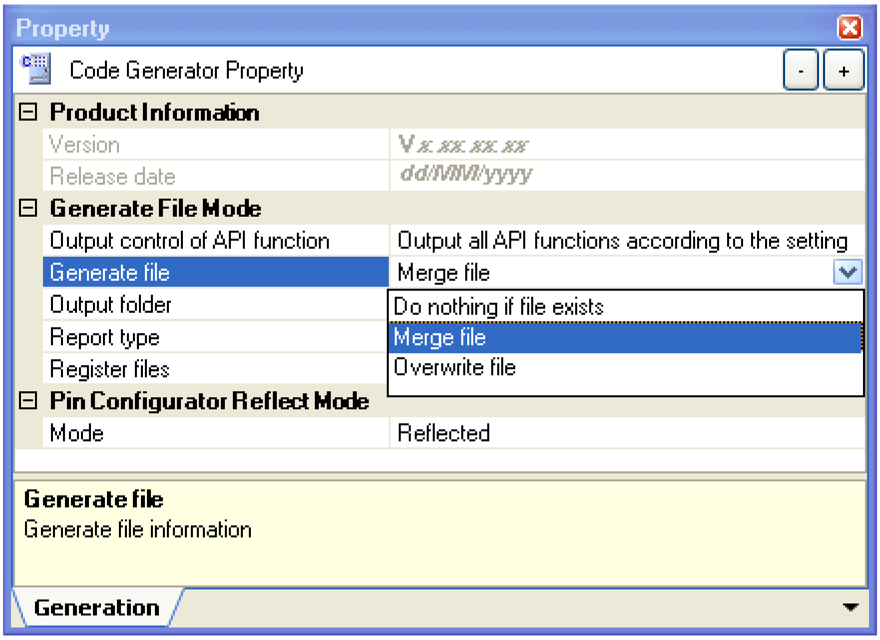


Remark To restore the default name of the API function defined by Code Generator, select [Default] from the context menu.

3.5.4 Change output mode

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

Figure 3-11. Change Output Mode



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

Table 3-5. Output Mode of Source Code

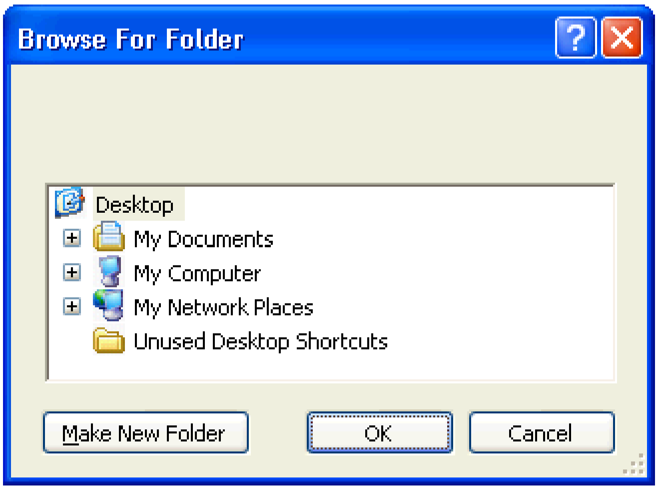
Output Mode	Outline
Do nothing if file exists	If a file with the same name exists, a new file will not be output.
Merge file	If a file with the same name exists, a new file is merged with the existing file. Only the section between "/* Start user code ... . Do not edit comment generated here */" and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

3.5.5 Change output destination folder

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

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

Figure 3-12. Change Output Destination Folder



3.6 Output Report Files

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

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

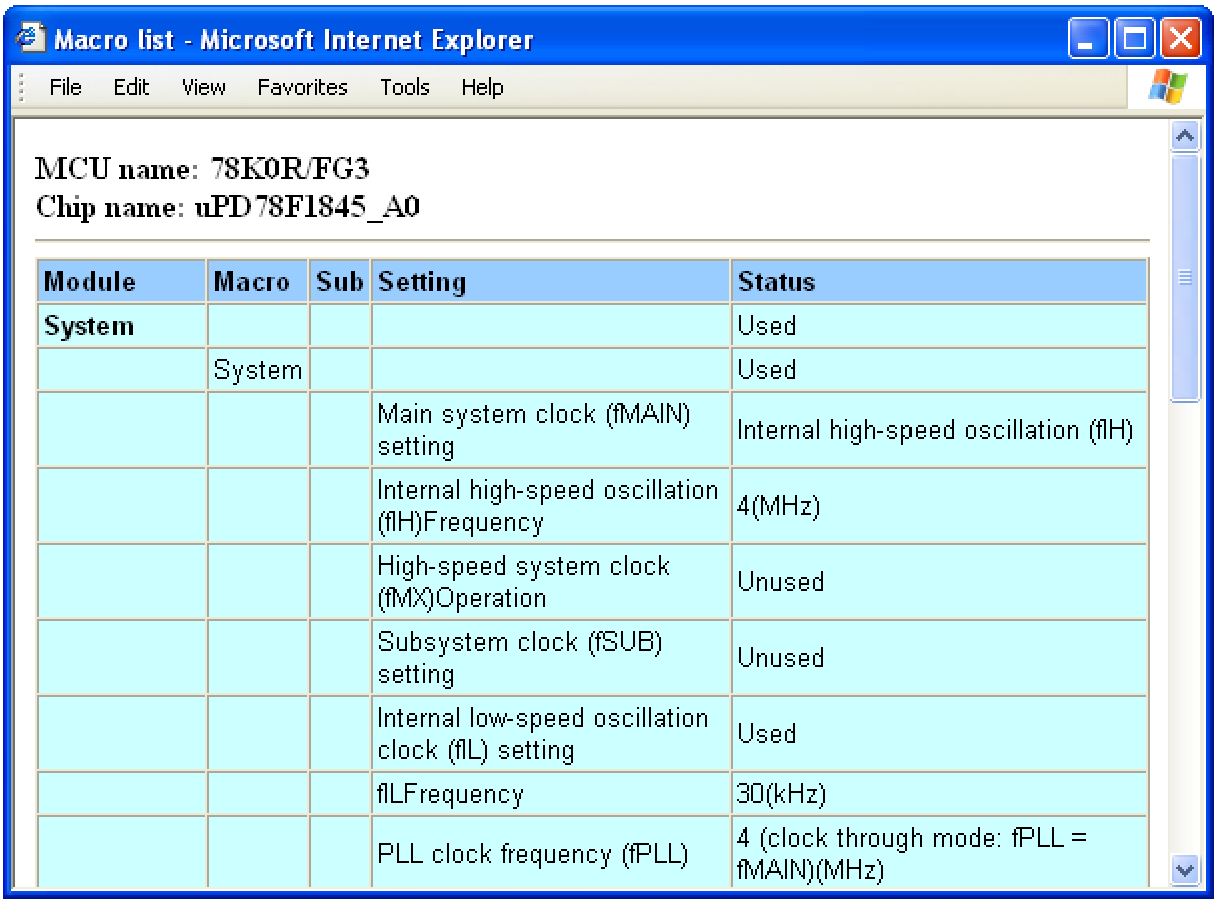
Remarks 1. You can only use "macro" or "function" as a name of the report file.

Table 3-6. Output Report Files

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

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

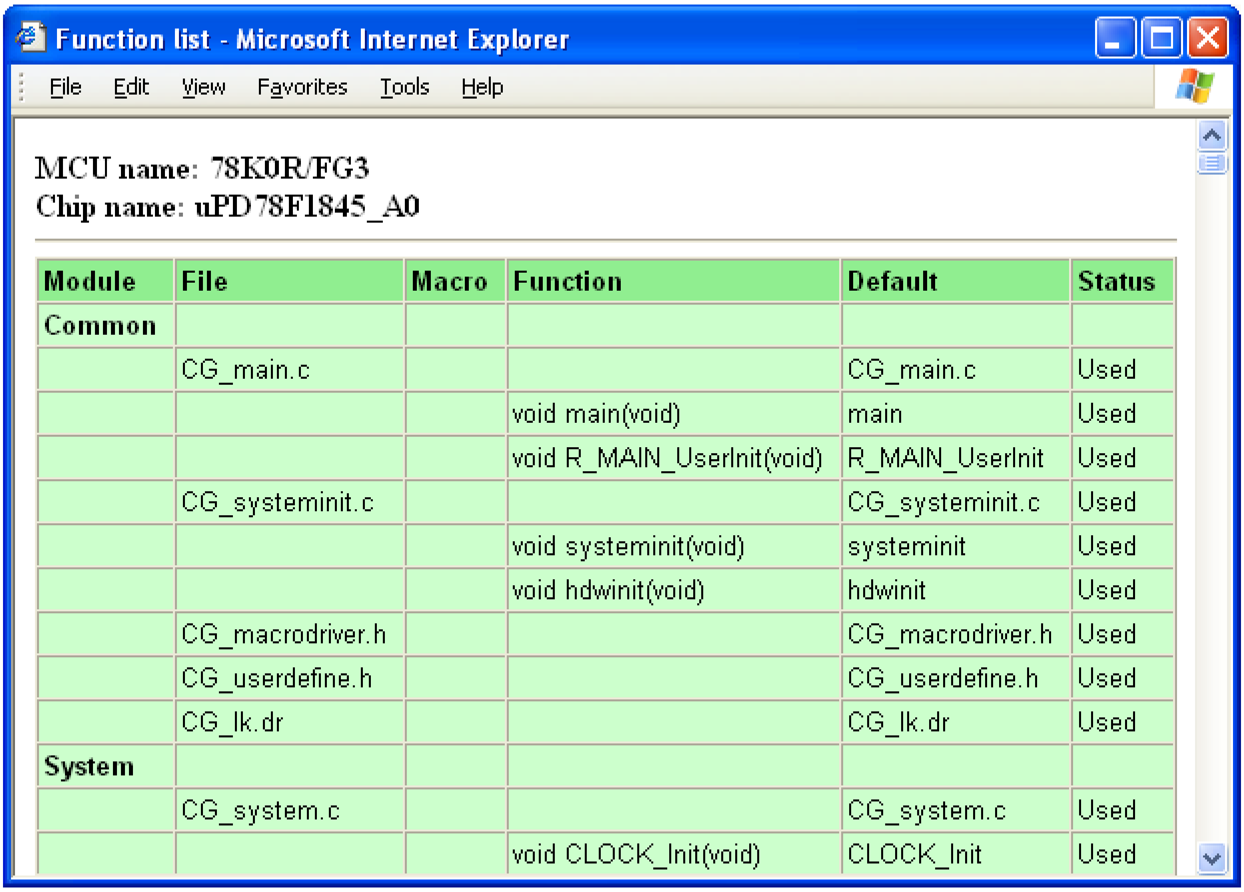
Figure 3-13. Output Example of Report File "macro"



MCU name: 78K0R/FG3
Chip name: uPD78F1845_A0

Module	Macro	Sub	Setting	Status
System				Used
	System			Used
			Main system clock (fMAIN) setting	Internal high-speed oscillation (fIH)
			Internal high-speed oscillation (fIH)Frequency	4(MHz)
			High-speed system clock (fMX)Operation	Unused
			Subsystem clock (fSUB) setting	Unused
			Internal low-speed oscillation clock (fIL) setting	Used
			fILFrequency	30(kHz)
			PLL clock frequency (fPLL)	4 (clock through mode: fPLL = fMAIN)(MHz)

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



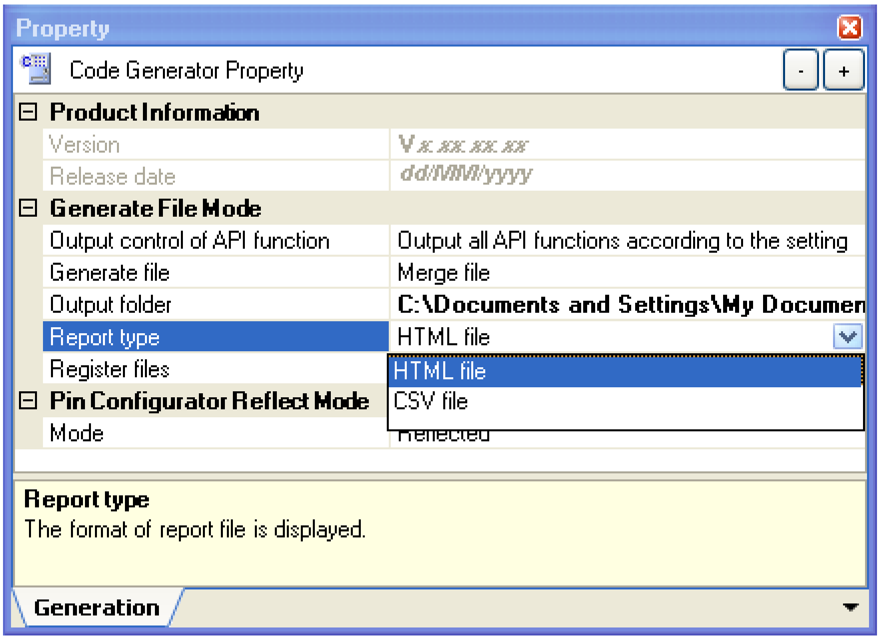
MCU name: 78K0R/FG3
Chip name: uPD78F1845_A0

Module	File	Macro	Function	Default	Status
Common					
	CG_main.c			CG_main.c	Used
			void main(void)	main	Used
			void R_MAIN_UserInit(void)	R_MAIN_UserInit	Used
	CG_systeminit.c			CG_systeminit.c	Used
			void systeminit(void)	systeminit	Used
			void hdwinit(void)	hdwinit	Used
	CG_macrodriver.h			CG_macrodriver.h	Used
	CG_userdefine.h			CG_userdefine.h	Used
	CG_lk.dr			CG_lk.dr	Used
System					
	CG_system.c			CG_system.c	Used
			void CLOCK_Init(void)	CLOCK_Init	Used

3.6.1 Change output format

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

Figure 3-15. Change Output Format



Remark Output format is selected from the following two types.

Table 3-7. Output Mode of Source Code

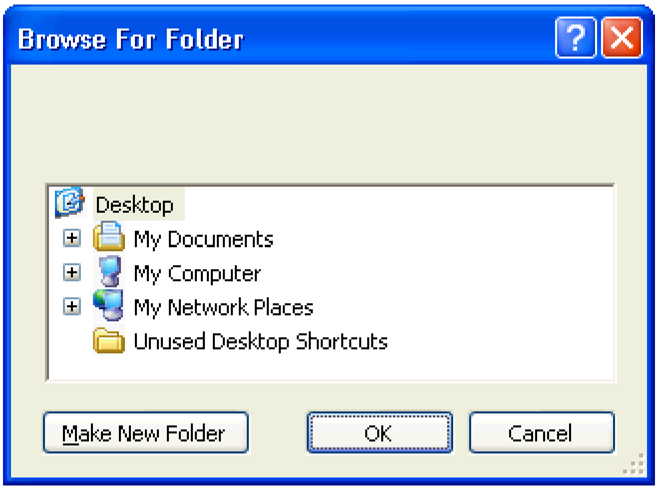
Report Type	Outline
HTML file	Outputs a report file in HTML format.
CSV file	Outputs a report file in CSV format.

3.6.2 Change output destination

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

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

Figure 3-16. Change Output Destination



APPENDIX A WINDOW REFERENCE

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

A.1 Description

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

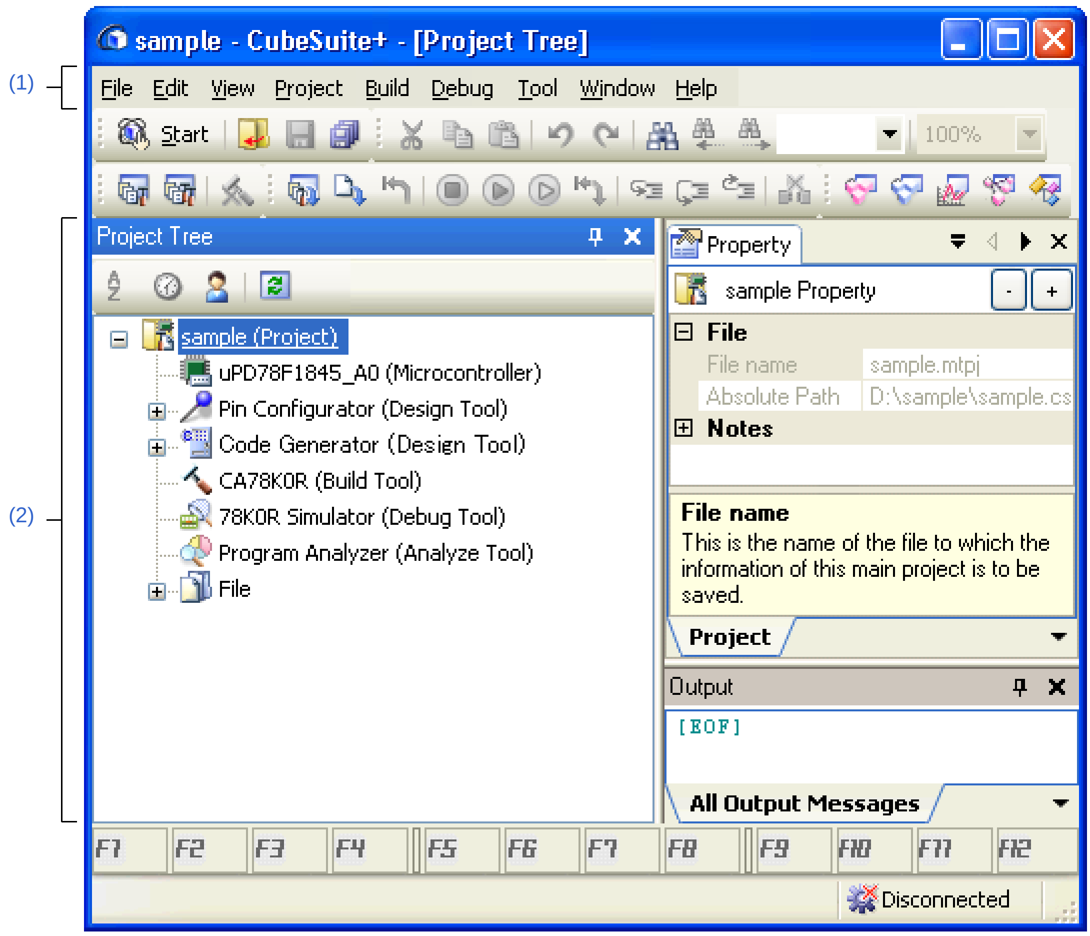
Table A-1. Window/Panel/Dialog Box List

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

Main window

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

Figure A-1. Main Window



The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Menu bar

This area consists of the following menu items.

(a) [File] menu

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

(c) [Help] menu

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

(2) Panel display area

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

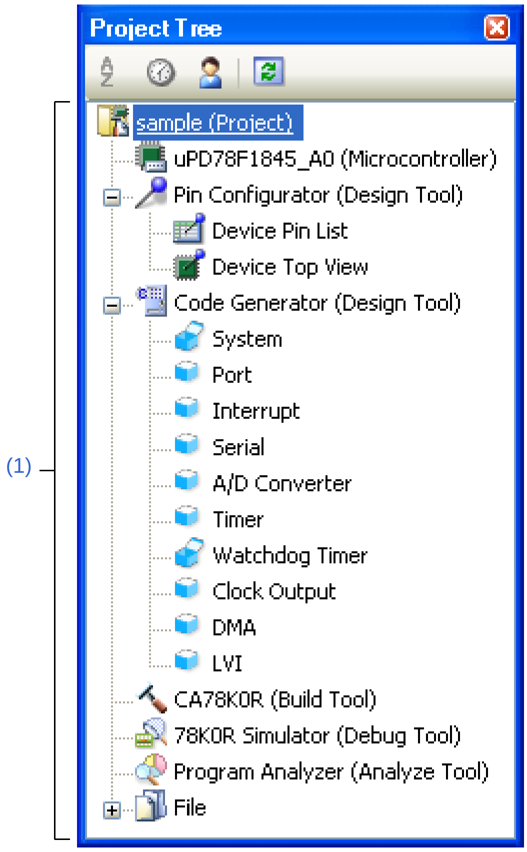
See the following sections for details on this area.

- Project Tree panel
- Property panel
- Device Pin List panel
- Device Top View panel
- Code Generator panel
- Code Generator Preview panel
- Output panel

Project Tree panel

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

Figure A-2. Project Tree Panel



The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Project tree area

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

(a) Pin Configurator (Design Tool)

This node consists of the following pin nodes.

Device Pin List	Opens the Device Pin List panel for entering information on the pins of the microcontroller.

Device Top View	Opens the Device Top View panel that displays the information entered in the Device Pin List panel.

(b) Code Generator (Design Tool)

This node consists of the following peripheral function nodes.

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

System	Opens the [System] for configuring the information necessary to control the functions of clock generator, on-chip debug function and etc. provided by the microcontroller.
External Bus	Opens the [External Bus] for configuring the information necessary to control the functions of external bus interface (functions to connect an external bus to the area other than the built-in ROM, RAM or SFR) 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.
Operational Amplifier	Opens the [Operational Amplifier] for configuring the information necessary to control the functions of operational amplifier provided by the microcontroller.
Comparator/PGA	Opens the [Comparator/PGA] for configuring the information necessary to control the functions of comparator/programmable gain amplifier 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 counter provided by the microcontroller.
Clock Output	Opens the [Clock Output] for configuring the information necessary to control the functions of clock output controller 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.
LCD Controller/Driver	Opens the [LCD Controller/Driver] for configuring the information necessary to control the function of LCD controller/driver provided by the microcontroller.
DMA	Opens the [DMA] for configuring the information necessary to control the functions of DMA (Direct Memory Access) controller provided by the microcontroller.

LVI	Opens the [LVI] for configuring the information necessary to control the functions of low-voltage detector provided by the microcontroller.

(c) Icons

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

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

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

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

[Context menu]

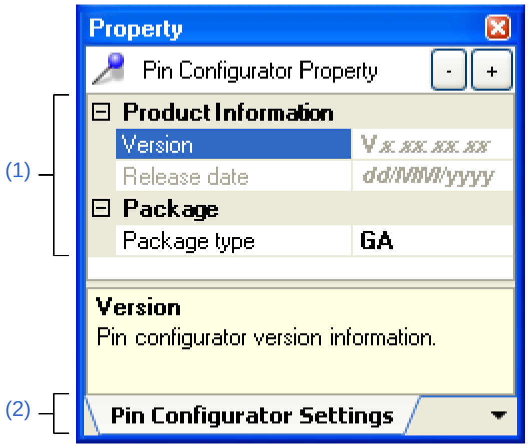
The following context menu items are displayed by right clicking the mouse.

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

Property panel

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

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



The following items are explained here.

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

[How to open]

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

Remarks 1. If this panel is already open, selecting a different node ([Pin Configurator (Design Tool)], [Device Pin List], [Device Top View], [Code Generator (Design Tool)] or peripheral function node "[System], [Port], etc.") in the Project Tree panel changes the content displayed in the Detail information display/change area and eaplanation area accordingly.

2. If this panel is already open, pressing a different peripheral function button "  ,  , 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 ([Pin Configurator (Design Tool)], [Device Pin List], [Device Top View], [Code Generator (Design Tool)] or peripheral function node "[System], [Port], etc.") selected in the Project Tree panel, the peripheral function button " , , etc." pressed in the Code Generator panel, or the file selected in the Code Generator Preview panel.

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

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

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

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

(2) Tab selection area

Categories for the display of the detailed information are changed when each tab is selected.

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

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

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

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

[Context menu]

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

(1) While the item is being edited

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

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

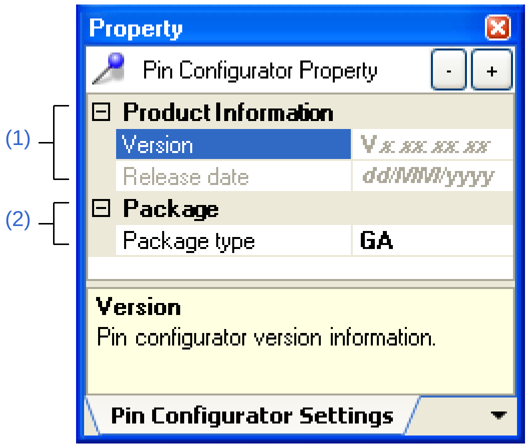
(2) While the item is not being edited

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

[Pin Configurator Settings] tab

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

Figure A-4. [Pin Configurator Settings] Tab



The following items are explained here.

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

[How to open]

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

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

[Description of each area]

(1) [Product Information] category

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

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

(2) [Package] category

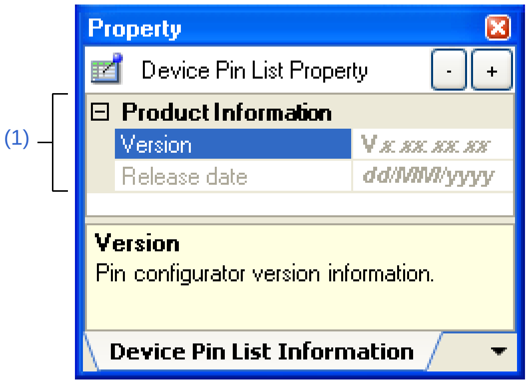
Change the shape (Package type) and settings of the microcontroller to display as the device top view in the Device Top View panel.

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

[Device Pin List Information] tab

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

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



The following items are explained here.

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

[How to open]

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

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

[Description of each area]

(1) [Product Information] category

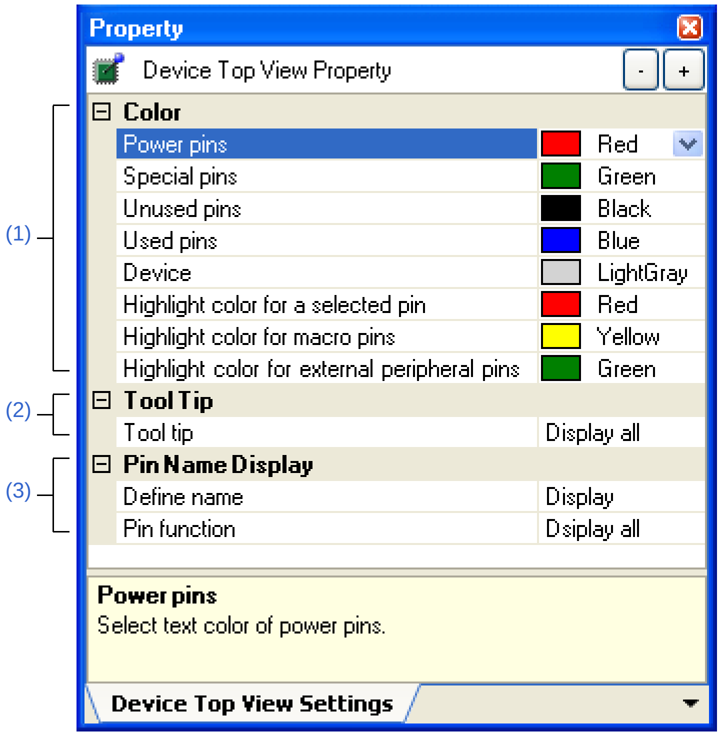
This area displays product information (Version and Release date) on Pin Configurator.

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

[Device Top View Settings] tab

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

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



The following items are explained here.

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

[How to open]

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

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

[Description of each area]

(1) [Color] category

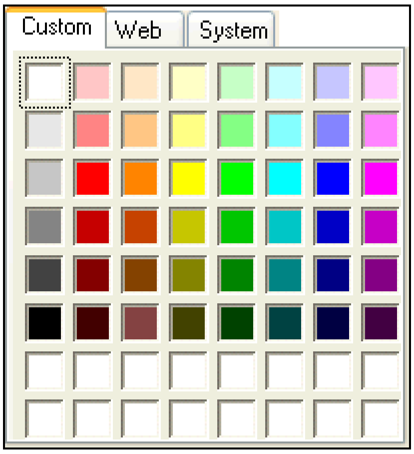
Select the display colors to differentiate the pin groups (Power pins, Special pins, etc.) in the device top view.

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

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

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

Figure A-7. Color Palette



(2) [Tool Tip] category

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

<table>
<tr><td rowspan="5">Tool tip</td><td colspan="2">Selects whether to display a tooltip with information about a pin when the mouse cursor is moved over the pin in the device top view panel.</td></tr>
<tr><td>Display all</td><td>Displays the "Description", "Recommend Connection for Unused", and "Attention" strings for the device pin list.</td></tr>
<tr><td>Description / recommended connection for unused pin only</td><td>Displays the "Description", and "Recommend Connection for Unused" strings for the device pin list.</td></tr>
<tr><td>Attention only</td><td>Displays the "Attention" string for the device pin list.</td></tr>
<tr><td>Not display</td><td>Hides tooltips when the mouse cursor hovers over a pin.</td></tr>
</table>

(3) [Pin Name Display] category

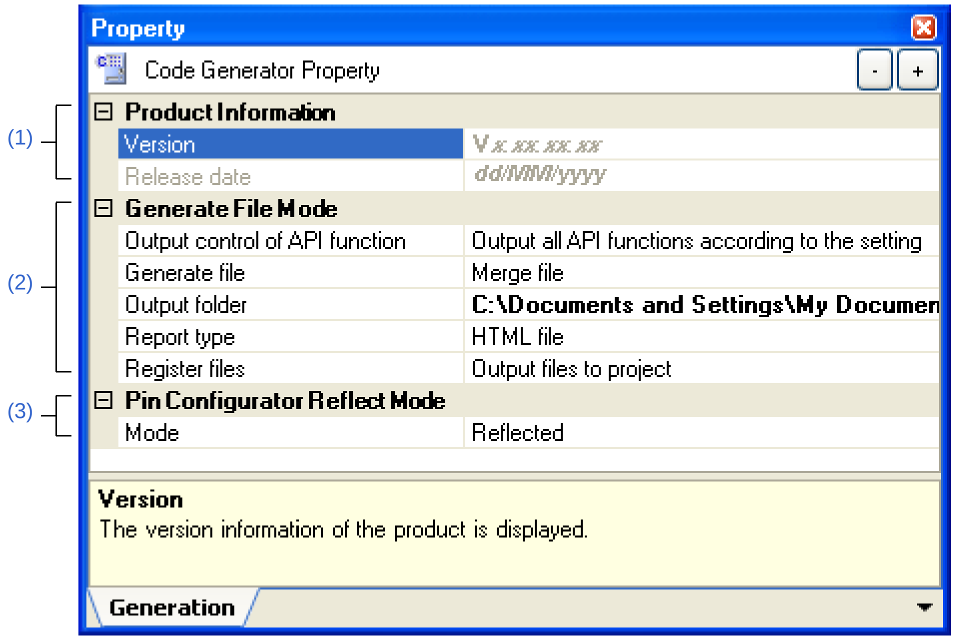
Select whether to display additional information about the pin in the device top view.

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

[Generation] tab

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

Figure A-8. [Generation] Tab



The following items are explained here.

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

[How to open]

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

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

[Description of each area]

(1) [Product Information] category

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

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

(2) [Generate File Mode] category

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

Output control of API function	Views or Selects the type of output API functions (all API functions or only initialization API functions) when the Generate Code button is pressed.	
	Output all API functions according to the setting	Outputs all API functions.
	Output only initialization API function	Outputs only initialization API functions.
Generate file	Views or selects the operation mode applied when the Generate Code button is pressed. Operation mode applied when you select [File] menu >> [Save Code Generator Report] is fixed to "Overwrite file".	
	Do nothing if file exists	If a file with the same name exists, a new file will not be output.
	Merge file	If a file with the same name exists, a new file is merged with the existing file. Only the section between "/* Start user code ... . Do not edit comment generated here */" and "/* End user code. Do not edit comment generated here */" will be merged.
	Overwrite file	If a file with the same name exists, the existing file is overwritten by a new file.
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

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

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

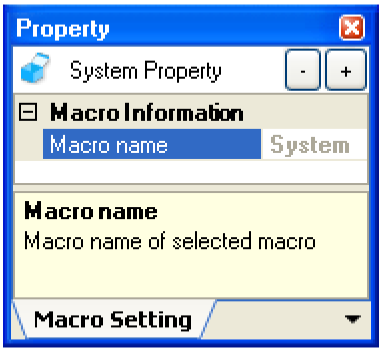
Remark If "Not reflected" is selected, then the Reflect in Pin button will be grayed out (deselected).

[Macro Setting] tab

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

Figure A-9. [Macro Setting] Tab

(1)



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 "[System], [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 "[System], [Port], etc.", and then select [Property] from the context menu.

Remarks 1. If this panel is already open, selecting a different peripheral function node "[System], [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

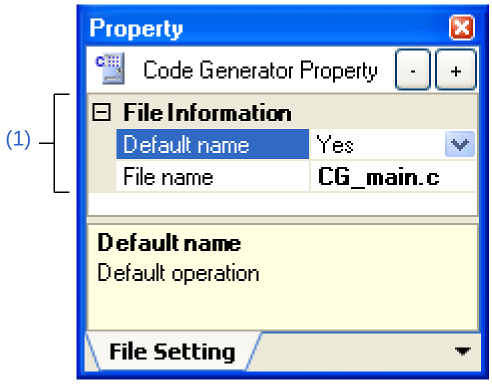
This area allows you to view the information (Macro name) on and change the setting for the peripheral function node "[System], [Port], etc." selected in the Project Tree panel, or the peripheral function button pressed in the Code Generator panel.

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

[File Setting] tab

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

Figure A-10. [File Setting] Tab



The following items are explained here.

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

[How to open]

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

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

[Description of each area]

(1) [File Information] category

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

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

Device Pin List panel

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

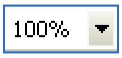
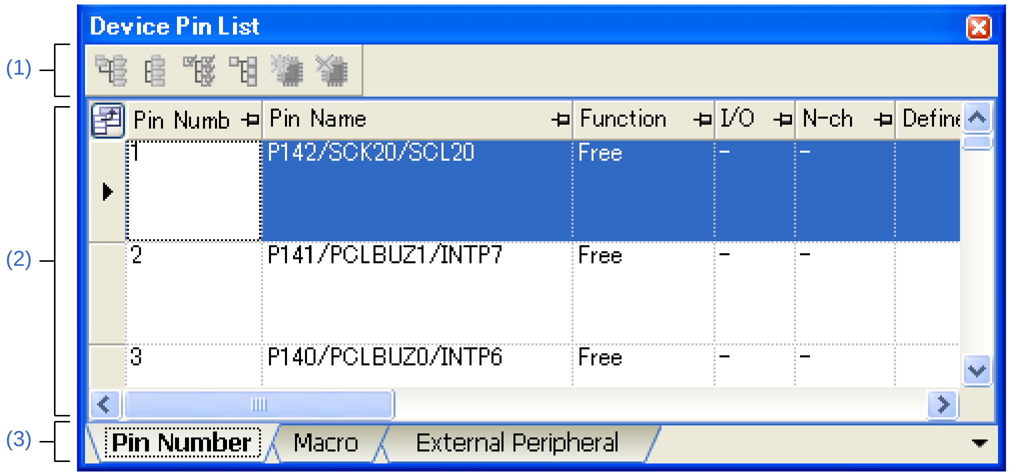
Remark The Device pin list area can be zoomed in and out by  in the tool bar, or by operating the mouse wheel while holding down the [Ctrl] key.

Figure A-11. Device Pin List Panel



The following items are explained here.
- [How to open]
- [Description of each area]
- [[File] menu (Device Pin List panel-dedicated items)]
- [[Help] menu (Device Pin List panel-dedicated items)]

[How to open]

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

[Description of each area]

(1) Toolbar

This area consists of the following buttons.

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

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

Remarks 1. Click the  button to add the information in question as a choice in the "External Parts" column of the [Macro] tab and the [Pin Number] tab.

2. Click the  button to remove the external peripheral component in question from the Device top view area if the Device Top View panel.

(2) Device pin list area

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

(3) Tab selection area

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

This panel has the following tabs:

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

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

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

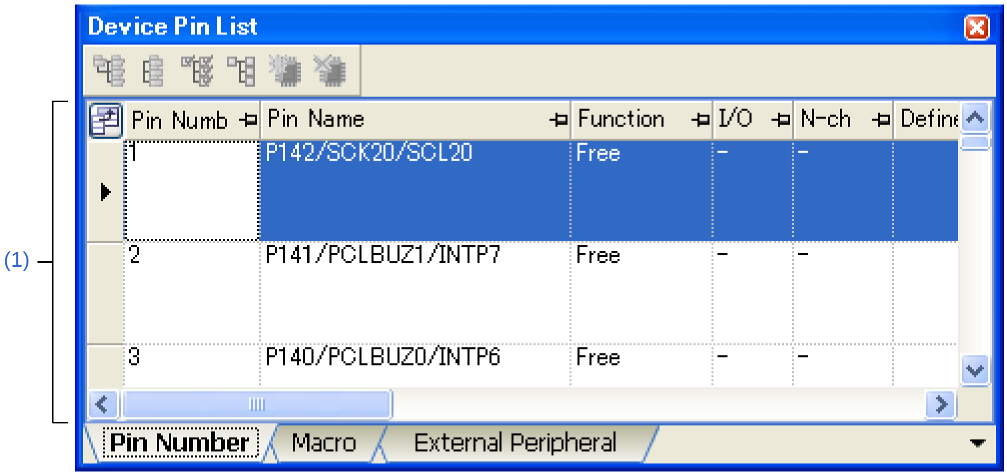
[[Help] menu (Device Pin List panel-dedicated items)]

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

[Pin Number] tab

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

Figure A-12. [Pin Number] Tab



The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Device pin list area

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

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

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

Remarks 1. You cannot add information in the "Pin Number" column, "Pin Name" column, "Description" column, "Recommend Connection for Unused" column and "Attention" column because they contain fixed information.

2. If the "Free" in the "Function" column is changed to a specific pin name, color of the corresponding pin in the Device Top View panel changes from the "color representing the unused pins" to the "color representing the used pins" selected by clicking [Device Top View Settings] tab >> [Color] in the Property panel.

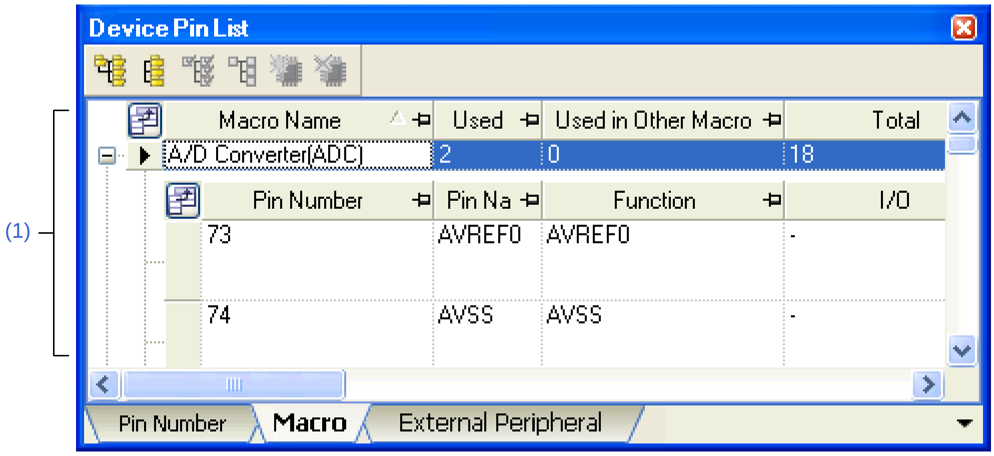
3. To move columns (change the display order) in the device pin list, drag and drop the desired column to the desired location.

4. To add the "user's own column", use the New Column dialog box which opens by pressing the [New Column...] button in the Column Chooser dialog box which opens by pressing the  button in the upper left corner of the device pin list.

[Macro] tab

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

Figure A-13. [Macro] Tab



The following items are explained here.
- [How to open]
- [Description of each area]

[How to open]

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

[Description of each area]

(1) Device pin list area

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

(a) First layer

The following are the columns comprising the device pin list.

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

(b) Second layer

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

Remarks 1. You cannot add information in the "Macro Name", "Total", "Used", "Used by other function", "Pin Number", "Pin Name", "Description", "Recommend Connection for Unused" and "Attention" columns because they contain fixed information.

2. If the "Free" in the "Function" column is changed to a specific pin name, color of the corresponding pin in the Device Top View panel changes from the "color representing the unused pins" to the "color representing the used pins" selected by clicking [Device Top View Settings] tab >> [Color] in the Property panel.

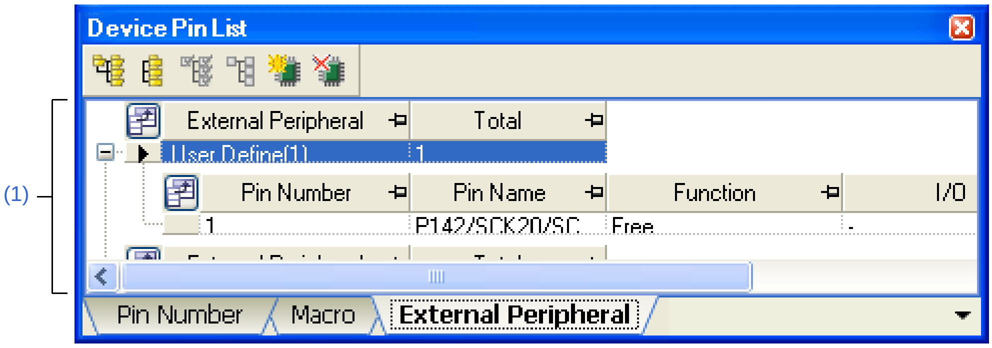
3. To move columns (change the display order) in the device pin list, drag and drop the desired column to the desired location.

4. To add the "user's own column", use the New Column dialog box which opens by pressing the [New Column...] button in the Column Chooser dialog box which opens by pressing the button in the upper left corner of the device pin list.

[External Peripheral] tab

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

Figure A-14. [External Peripheral] Tab



The following items are explained here.
- [How to open]
- [Description of each area]

[How to open]

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

[Description of each area]

(1) Device pin list area

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

(a) First layer

The following are the columns comprising the device pin list.

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

(b) Second layer

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

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

Remarks 1. You cannot add information in the "External Peripheral Name", "Connected Pins", "Pin Number", "Pin Name", "Description", "Recommend Connection for Unused" and "Attention" columns because they contain fixed information.

2. If the "Free" in the "Function" column is changed to a specific pin name, color of the corresponding pin in the Device Top View panel changes from the "color representing the unused pins" to the "color representing the used pins" selected by clicking [Device Top View Settings] tab >> [Color] in the Property panel.

3. To move columns (change the display order) in the device pin list, drag and drop the desired column to the desired location.

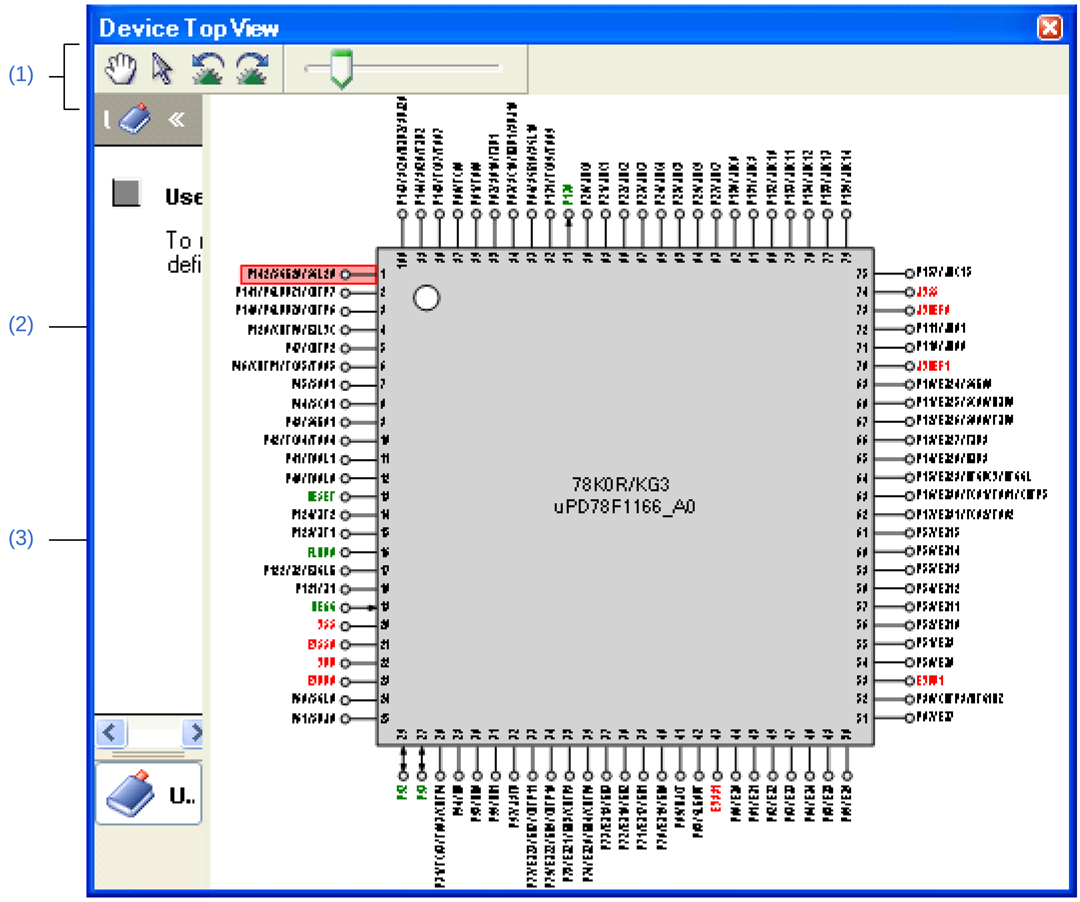
4. To add the "user's own column", use the New Column dialog box which opens by pressing the [New Column...] button in the Column Chooser dialog box which opens by pressing the button in the upper left corner of the device pin list.

Device Top View panel

This panel displays the information entered in the Device Pin List panel.

Remark The Device top view area can be zoomed in and out by 100% in the tool bar.

Figure A-15. Device Top View Panel



The following items are explained here.

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

[How to open]

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

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

[Description of each area]

(1) Toolbar

This area consists of the following buttons.

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

(2) [User Define] area

Drag and drop the button from this area to the Device top view area to creat and display an external peripheral controller.

(3) Device top view area

This area displays the pin assignment of the microcontroller.

Settings of the pin assignment are displayed using the colors specified by selecting [Device Top View Settings] tab >> [Color] in the Property panel.

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

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

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

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

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

[Context menu]

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

(1) When a pin is right clicked

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

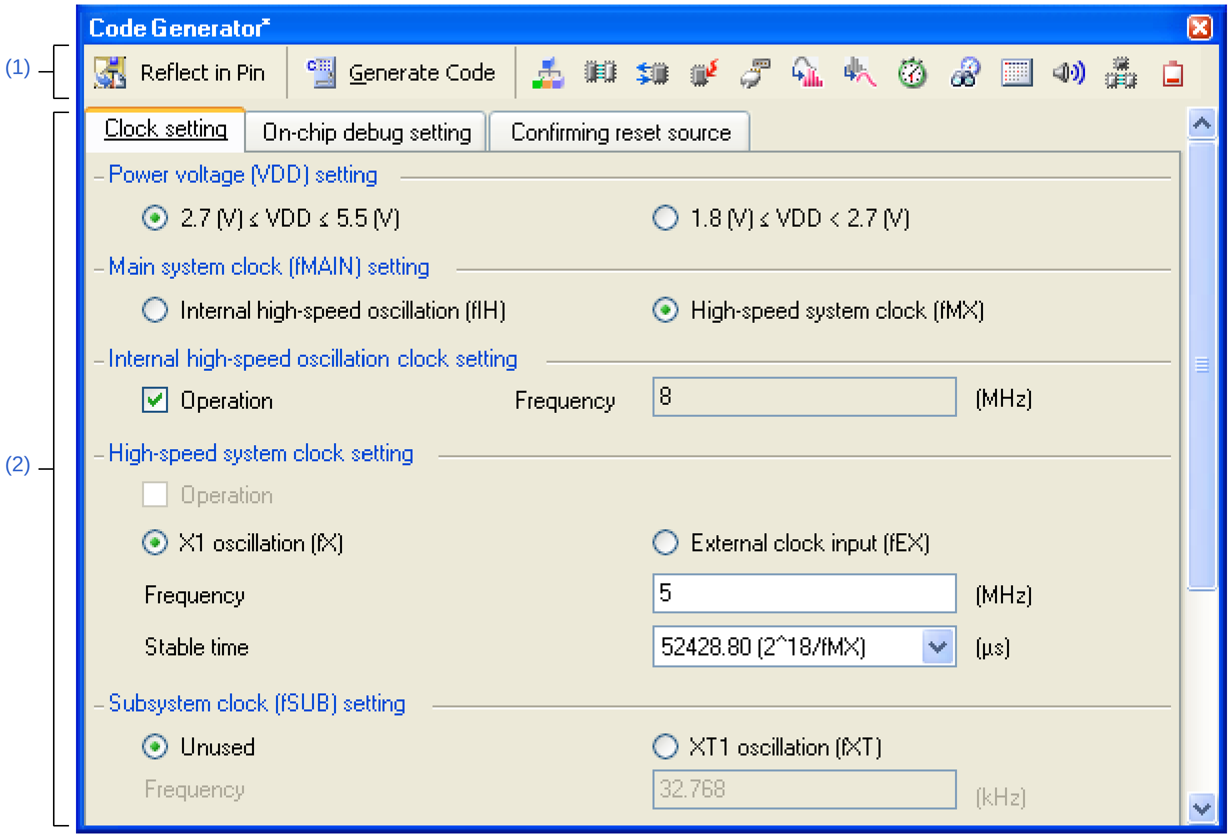
(2) When an external peripheral controller is right clocked

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

Code Generator panel

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

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



The following items are explained here.

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

[How to open]

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

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

[Description of each area]

(1) Toolbar

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

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

Reflect in Pin	Reflects settings made on this panel in the Device Pin List panel, and then output the changed contents to the Output panel. This button will be grayed out (disabled) if "Not reflected" is selected in the [PinPart Combination Mode] category of the [Generation] tab.
Generate Code	Outputs the source code (device driver program) to the folder specified by selecting [Generation] tab >> [Output folder] in the Property panel.
	Changes the content displayed in the Information setting area to the "[System] 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 "[External Bus] for configuring the information necessary to control the functions of external bus interface (functions to connect an external bus to the area other than the built-in ROM, RAM or SFR) 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 "[Operational Amplifier] for configuring the information necessary to control the functions of operational amplifier provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Comparator/PGA] for configuring the information necessary to control the functions of comparator/programmable gain amplifier 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 counter provided by the microcontroller".

	Changes the content displayed in the Information setting area to the "[Clock Output] for configuring the information necessary to control the functions of clock output controller 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 "[LCD Controller/Driver] for configuring the information necessary to control the function of LCD controller/driver provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[DMA] 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 "[LVI] for configuring the information necessary to control the functions of low-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.

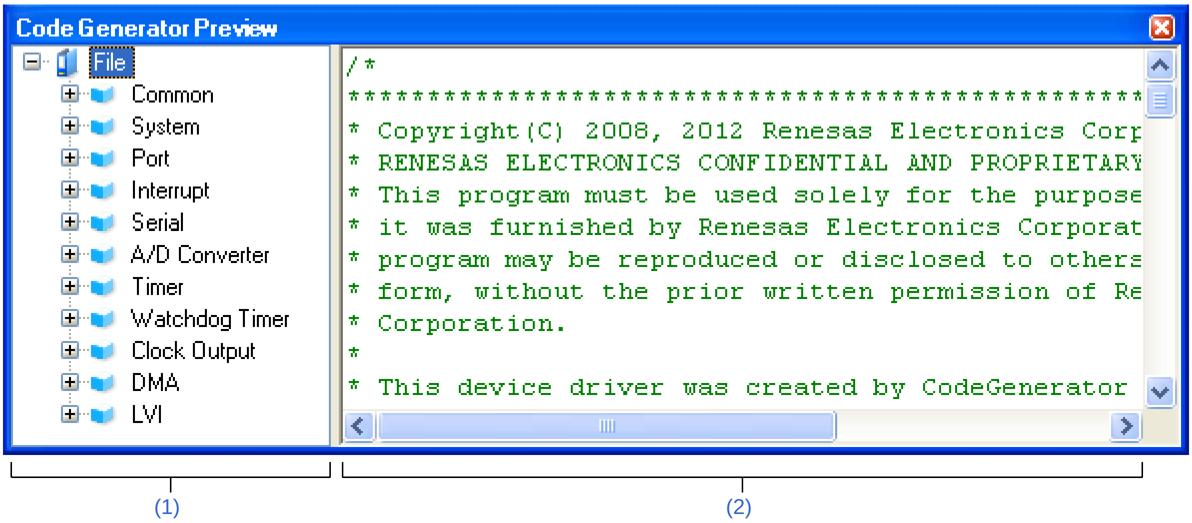
[[Help] menu (Code Generator panel-dedicated items)]

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-17. Code Generator Preview Panel



The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Preview tree

This area allows you to confirm or configure on a per-API function basis the setting that determines whether or not the source code (device driver program) is generated when the 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 Generate Code button on the Code Generator panel is pressed). For this reason, the source code displayed in this panel may not be the same as that would actually be generated.

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

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

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

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

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

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

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

[Context menu]

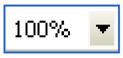
The following context menu items are displayed by right clicking the mouse.

Generate code	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 . This item will be grayed out (disabled) if the currently selected API function is not initialization API function, and "Output only initialization API function" is selected [Generation] tab >> [Output control of API function] in the Property panel.
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	Selecting this menu item 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

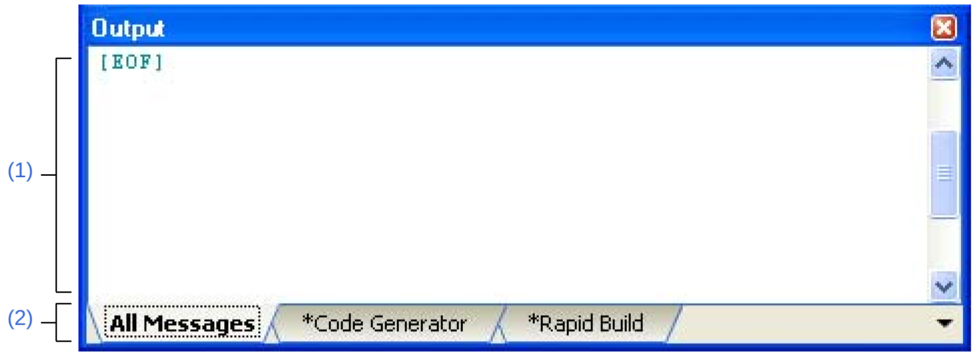
This panel is used to display operation logs for various components (design tool, build tool, debug tool, etc.) provided by CubeSuite+.

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

Remark The Message area can be zoomed in and out by 

in the tool bar, or by operating the mouse wheel while holding down the [Ctrl] key.

Figure A-18. Output Panel



The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Message area

The output messages of each tool are displayed.

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

Message Type	Display Example (Default)			Description
Normal message	ABCD abcd 0123	Character color	Black	Displayed with information notices.
		Background color	White	
Warning message	ABCD abcd 0123	Character color	Bule	Displayed with warnings about operations.
		Background color	Standard color	
Error message	ABCD abcd 0123	Character color	Red	Displayed when there is a critical error, or when execution is not possible due to a operational mistake.
		Background color	Light gray	

(2) Tab selection area

Select the tab that indicates the origin of message.
The following tabs are available for the debug tool.

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

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

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

Save Output-*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.

[Context menu]

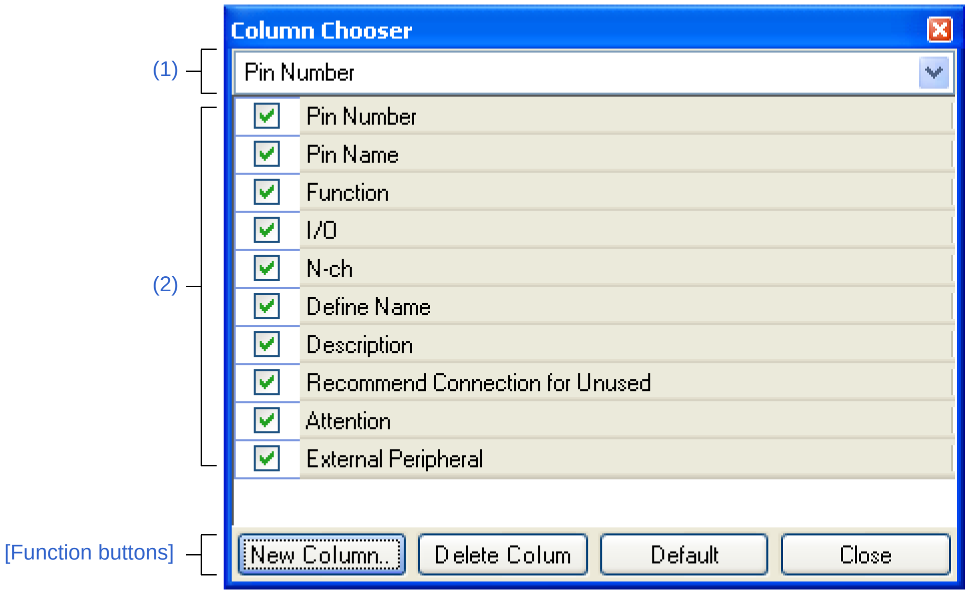
The following context menu items are displayed by right clicking the mouse.

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.

Column Chooser dialog box

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

Figure A-19. Column Chooser Dialog Box



The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Operational object selection area

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

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

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

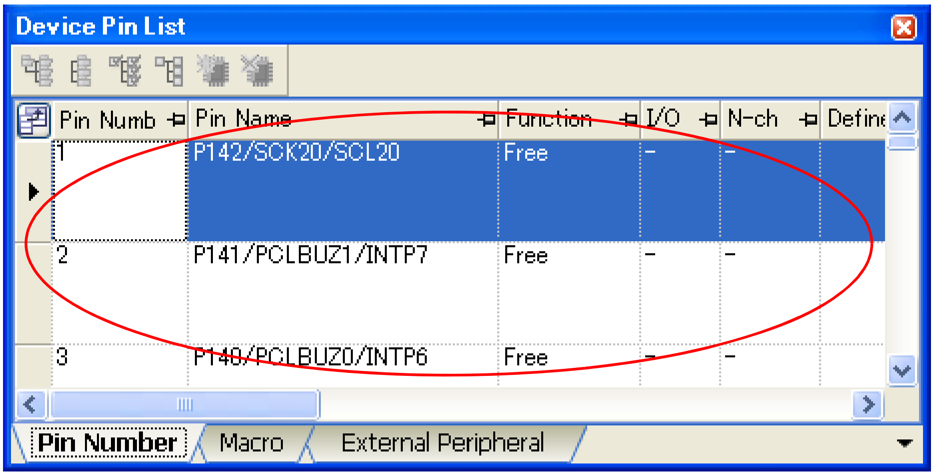


Figure A-21. Operational Object ([Macro] Tab: First Layer)

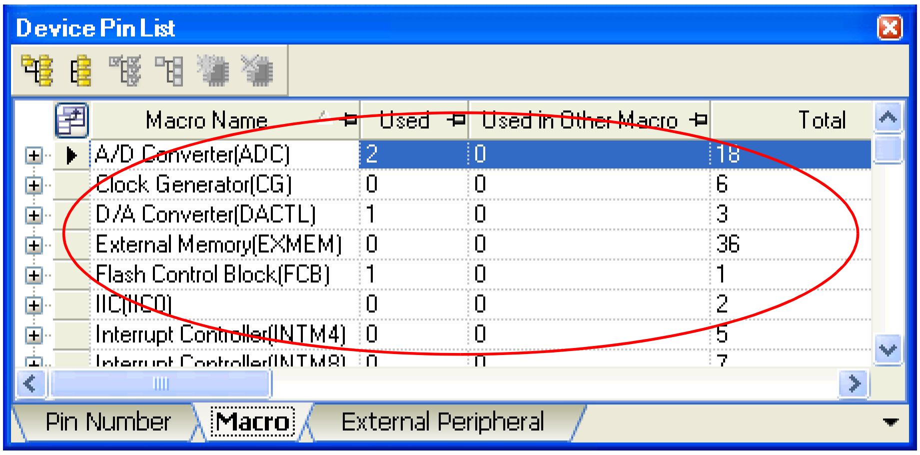


Figure A-22. Operational Object ([Macro] Tab: Second Layer)

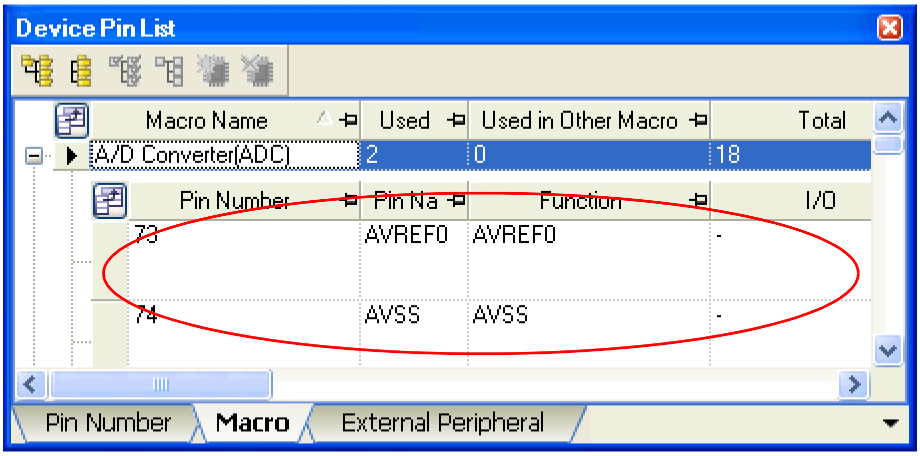


Figure A-23. Operational Object ([External Peripheral] Tab: First Layer)

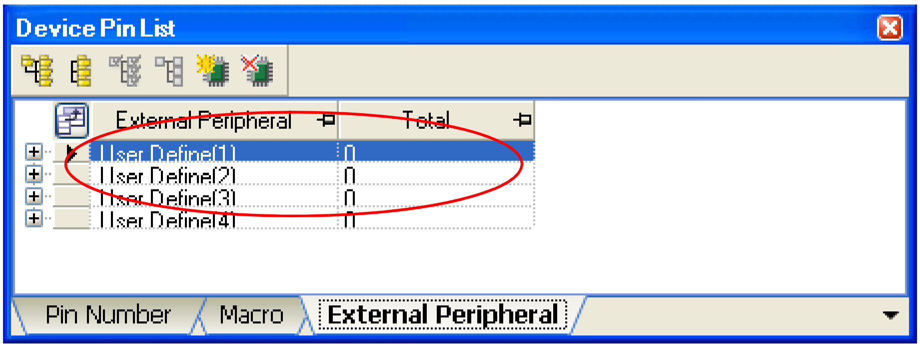
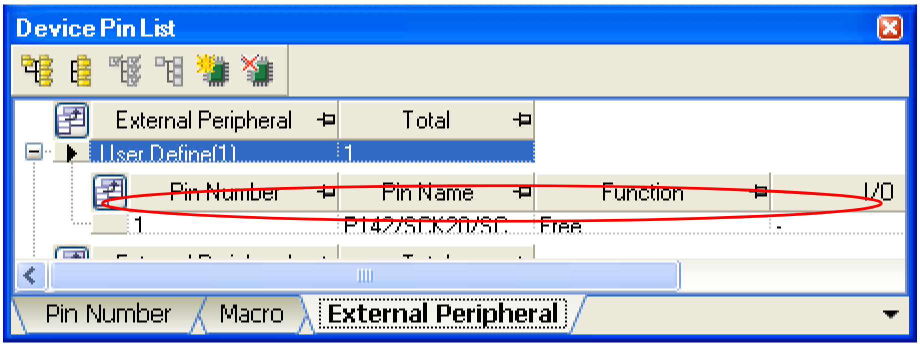


Figure A-24. Operational Object ([External Peripheral] Tab: Second Layer)



(2) Displayed item selection area

Select whether or not to display the item selected in the Operational object selection area in the device pin list.

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

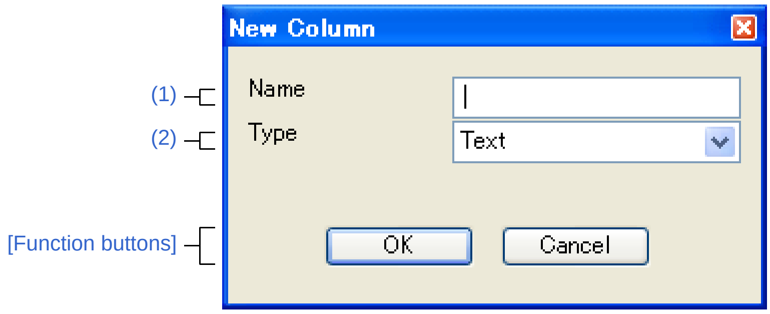
[Function buttons]

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

New Column dialog box

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

Figure A-25. New Column Dialog Box



The following items are explained here.
- [How to open]
- [Description of each area]
- [Function buttons]

[How to open]

- Click the [New Column...] button in the Column Chooser dialog box.

[Description of each area]

(1) [Name]

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

(2) [Type]

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

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

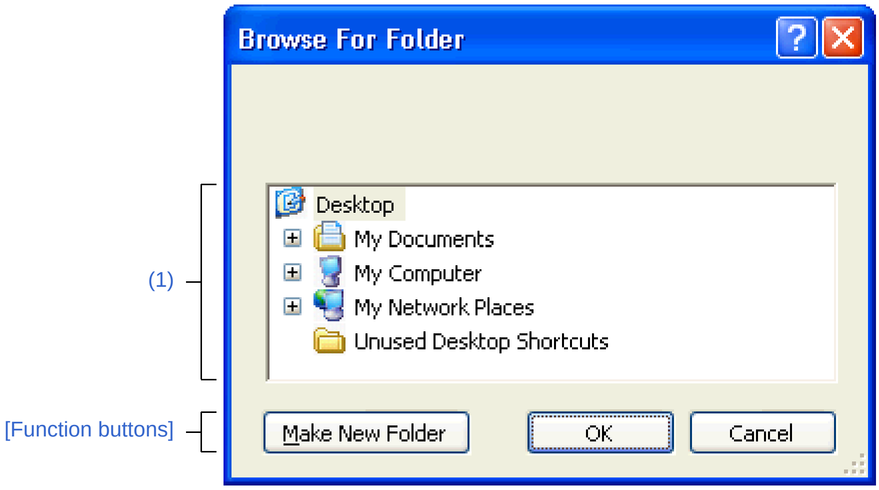
[Function buttons]

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

Browse For Folder dialog box

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

Figure A-26. Browse For Folder Dialog Box



The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) Folder location

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

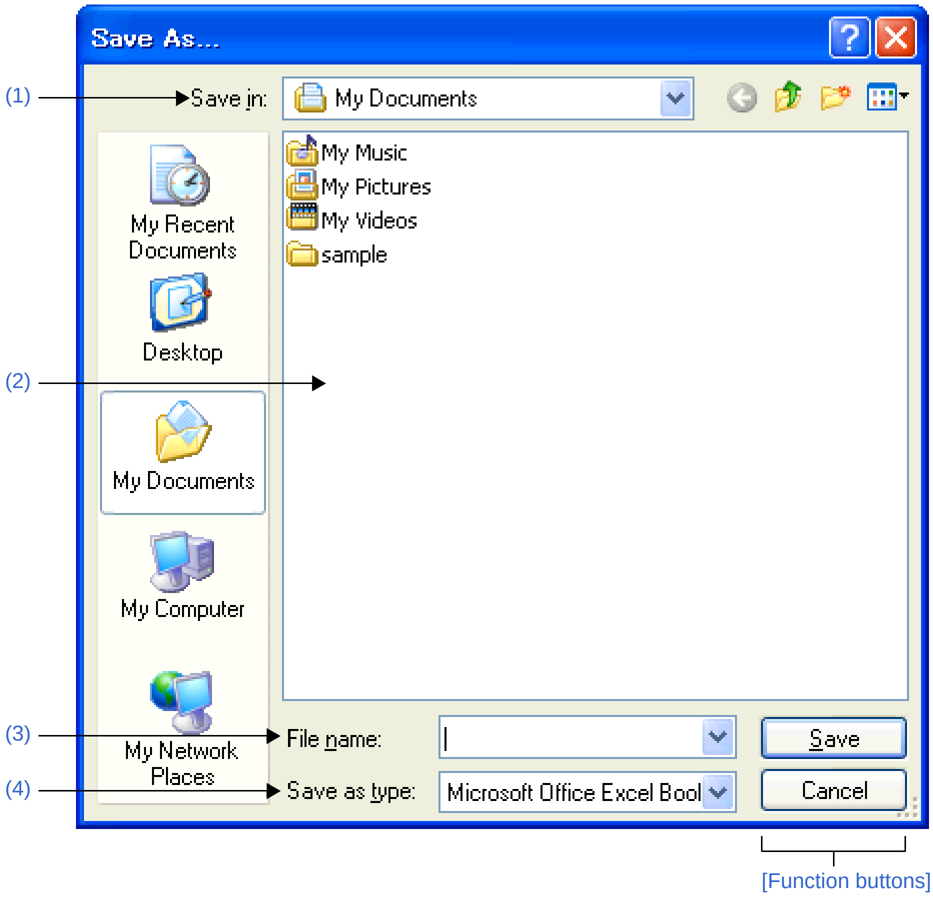
[Function buttons]

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

Save As dialog box

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

Figure A-27. Save As Dialog Box



The following items are explained here.

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

[How to open]

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

[Description of each area]

(1) [Save in]

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

(2) List of files

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

(3) [File name]

Specify the name of the file to be output.

(4) [Save as type]

Select the type of the file to be output.

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

[Function buttons]

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

APPENDIX B OUTPUT FILES

This appendix describes the files output by Code Generator.

B.1 Overview

Below is a list of files output by Code Generator.

Table B-1. File List

Unit of Output	File Name	Description
Peripheral function	CG_*PeripheralFunctionName*.c	Initial function, API function
	CG_*PeripheralFunctionName*_user.c	Interrupt function (MD_INT*xxx*), callback function
	CG_*PeripheralFunctionName*.h	Defines macros for assigning values to registers
Project	CG_main.c	main function, R_MAIN_UserInit function
	CG_systeminit.c	Call initial function of peripheral function Call CG_ReadResetSource
	CG_macrodriver.h	Defines common macros used by all source files
	CG_userdefine.h	Empty file (for user definitions)
	CG_lk.dr	Link directive

B.2 Output File

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

Table B-2. File List (Peripheral Function)

Peripheral Function	Source File Name	Names of API Functions Included
System	CG_system.c	CLOCK_Init CG_ChangeClockMode CG_ChangeFrequency CG_SelectPowerSaveMode CG_SelectStabTime
	CG_system_user.c	CLOCK_UserInit CG_ReadResetSource
	CG_system.h	-
External Bus	CG_bus.c	BUS_Init BUS_PowerOff
	CG_bus_user.c	BUS_UserInit
	CG_bus.h	-
Port	CG_port.c	PORT_Init PORT_ChangePmnInput PORT_ChangePmnOutput
	CG_port_user.c	PORT_UserInit
	CG_port.h	-
Interrupt	CG_int.c	INTP_Init

Peripheral Function	Source File Name	Names of API Functions Included
Interrupt	CG_int.c	KEY_Init INT_MaskableInterruptEnable INTPn_Disable INTPn_Enable KEY_Disable KEY_Enable
	CG_int_user.c	INTP_UserInit KEY_UserInit MD_INTP*n* MD_INTKR
	CG_int.h	-
Serial	CG_serial.c	SAUm_Init SAUm_PowerOff UARTn_Init UARTn_Start UARTn_Stop UARTn_SendData UARTn_ReceiveData CSImn_Init CSImn_Start CSImn_Stop CSImn_SendData CSImn_ReceiveData CSImn_SendReceiveData IICmn_Init IICmn_Stop IICmn_MasterSendStart IICmn_MasterReceiveStart IICmn_StartCondition IICmn_StopCondition UARTFn_Init UARTFn_PowerOff UARTFn_Start UARTFn_Stop UARTFn_SendData UARTFn_ReceiveData UARTFn_SetComparisonData UARTFn_DataComparisonEnable UARTFn_DataComparisonDisable IICA_Init IICA_PowerOff IICA_Stop IICA_MasterSendStart IICA_MasterReceiveStart IICA_StopCondition IICA_SlaveSendStart IICA_SlaveReceiveStart

Peripheral Function	Source File Name	Names of API Functions Included
Serial	CG_serial.c	IICn_Init IICn_Stop IICn_MasterSendStart IICn_MasterReceiveStart IICn_SlaveSendStart IICn_SlaveReceiveStart
	CG_serial_user.c	SAUm_UserInit UARTn_SendEndCallback UARTn_ReceiveEndCallback UARTn_SoftOverRunCallback UARTn_ErrorCallback CSImn_SendEndCallback CSImn_ReceiveEndCallback CSImn_ErrorCallback IICmn_MasterSendEndCallback IICmn_MasterReceiveEndCallback IICmn_MasterErrorCallback UARTFn_SendEndCallback UARTFn_ReceiveEndCallback UARTFn_SoftOverRunCallback UARTFn_ExpBitCetectCallback UARTFn_IDMatchCallback UARTFn_ErrorCallback IICA_UserInit IICA_MasterSendEndCallback IICA_MasterReceiveEndCallback IICA_MasterErrorCallback IICA_SlaveSendEndCallback IICA_SlaveReceiveEndCallback IICA_SlaveErrorCallback IICA_GetStopConditionCallback IICn_UserInit IICn_MasterSendEndCallback IICn_MasterReceiveEndCallback IICn_MasterErrorCallback IICn_SlaveSendEndCallback IICn_SlaveReceiveEndCallback IICn_SlaveErrorCallback IICn_GetStopConditionCallback MD_INTSR*n* MD_INTSRE*n* MD_INTST*n* MD_INTCSI*mn* MD_INTIIC*mn* MD_INTLT*n* MD_INTLR*n* MD_INTLS*n* MD_INTIIC*n*

Peripheral Function	Source File Name	Names of API Functions Included
Serial	CG_serial_user.c	MD_INTIICA
	CG_serial.h	-
Operational Amplifier	CG_opamp.c	OPAMP_Init AMPn_Start AMPn_Stop
	CG_opamp_user.c	OPAMP_UserInit
	CG_opamp.h	-
Comparator/PGA	CG_cmppga.c	CMPPGA_Init CMPPGA_PowerOff CMPPGA_Start CMPPGA_Stop CMPPGA_ChangeCMPnRefVoltage CMPPGA_ChangePGAFactor
	CG_cmppga_user.c	CMPPGA_UserInit MD_INTCMP*n*
	CG_cmppga.h	-
A/D Converter	CG_ad.c	AD_Init AD_PowerOff AD_ComparatorOn AD_ComparatorOff AD_Start AD_Stop AD_SelectADChannel AD_Read AD_ReadByte
	CG_ad_user.c	AD_UserInit MD_INTAD
	CG_ad.h	-
D/A Converter	CG_da.c	DA_Init DA_PowerOff DAn_Start DAn_Stop DAn_SetValue DAn_Set8BitsValue DAn_Set12BitsValue
	CG_da_user.c	DA_UserInit
	CG_da.h	-
Timer	CG_timer.c	TAUm_Init TAUm_PowerOff TAUm_Channeln_Start TAUm_Channeln_Stop TAUm_Channeln_ChangeCondition TAUm_Channeln_ChangeTimerCondition TAUm_Channeln_GetPulseWidth

<table>
<tr><th>Peripheral Function</th><th>Source File Name</th><th>Names of API Functions Included</th></tr>
<tr><td rowspan="3">Timer</td><td>CG_timer.c</td><td>TAUm_Channeln_ChangeDuty
TAUm_Channeln_SoftWareTriggerOn</td></tr>
<tr><td>CG_timer_user.c</td><td>TAUm_UserInit
MD_INTTMmn</td></tr>
<tr><td>CG_timer.h</td><td>-</td></tr>
<tr><td rowspan="3">Watchdog Timer</td><td>CG_wdt.c</td><td>WDT_Init
WDT_Restart</td></tr>
<tr><td>CG_wdt_user.c</td><td>WDT_UserInit
MD_INTWDTI</td></tr>
<tr><td>CG_wdt.h</td><td>-</td></tr>
<tr><td rowspan="3">Real-time Clock</td><td>CG_rtc.c</td><td>RTC_Init
RTC_PowerOff
RTC_CounterEnable
RTC_CounterDisable
RTC_SetHourSystem
RTC_CounterSet
RTC_CounterGet
RTC_ConstPeriodInterruptEnable
RTC_ConstPeriodInterruptDisable
RTC_AlarmEnable
RTC_AlarmDisable
RTC_AlarmSet
RTC_AlarmGet
RTC_IntervalStart
RTC_IntervalStop
RTC_IntervalInterruptEnable
RTC_IntervalInterruptDisable
RTC_RTC1HZ_OutputEnable
RTC_RTC1HZ_OutputDisable
RTC_RTCCL_OutputEnable
RTC_RTCCL_OutputDisable
RTC_RTCDIV_OutputEnable
RTC_RTCDIV_OutputDisable
RTC_ChangeCorrectionValue</td></tr>
<tr><td>CG_rtc_user.c</td><td>RTC_UserInit
RTC_ConstPeriodInterruptCallback
RTC_AlarmInterruptCallback
MD_INTRTC
MD_INTRTCI</td></tr>
<tr><td>CG_rtc.h</td><td>-</td></tr>
<tr><td rowspan="2">Clock Output</td><td>CG_pcl.c</td><td>PCL_Init
PCL_Start
PCL_Stop
PCL_ChangeFreq</td></tr>
<tr><td>CG_pcl_user.c</td><td>PCL_UserInit</td></tr>
</table>

Peripheral Function	Source File Name	Names of API Functions Included
Clock Output	CG_pcl.h	-
Clock Output/Buzzer Output	CG_pclbuz.c	PCLBUZn_Init PCLBUZn_Start PCLBUZn_Stop PCLBUZn_ChangeFreq
	CG_pclbuz_user.c	PCLBUZn_UserInit
	CG_pclbuz.h	-
LCD Controller/Driver	CG_lcd.c	LCD_Init LCD_DisplayOn LCD_DisplayOff LCD_VoltageOn LCD_VoltageOff
	CG_lcd_user.c	LCD_UserInit
	CG_lcd.h	-
DMA	CG_dma.c	DMAn_Init DMAn_Enable DMAn_Disable DMAn_Hold DMAn_Restart DMAn_CheckStatus DMAn_SetData DMAn_SoftwareTriggerOn
	CG_dma_user.c	DMAn_UserInit MD_INTDMA*n*
	CG_dma.h	-
LVI	CG_lvi.c	LVI_Init LVI_InterruptModeStart LVI_ResetModeStart LVI_Stop LVI_SetLVILevel
	CG_lvi_user.c	LVI_UserInit MD_INTLVI
	CG_lvi.h	-

APPENDIX C API FUNCTIONS

This appendix describes the API functions output by Code Generator.

C.1 Overview

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

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

C.2 Output Function

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

Table C-1. API Function List

Peripheral Function	API Function Name	Function
System	CLOCK_Init	Performs initialization required to control the clock generator, on-chip debug, and etc. .
	CLOCK_UserInit	Performs user-defined initialization relating to the clock generator, on-chip debug, and etc. .
	CG_ReadResetSource	Performs processing in response to RESET signal.
	CG_ChangeClockMode	Changes the CPU clock/peripheral hardware clock.
	CG_ChangeFrequency	Changes the division ratio of the CPU clock/peripheral hardware clock.
	CG_SelectPowerSaveMode	Configures the CPU's standby function.
	CG_SelectStabTime	Configures the oscillation stabilization time of the X1 clock.
External Bus	BUS_Init	Performs initialization necessary to control external bus interface functions (functions to connect an external bus to areas other than onboard ROM, ROM and RAM).
	BUS_UserInit	Performs user-defined initialization relating to the external bus interface.
	BUS_PowerOff	Halts the clock supplied to the external bus interface.
Port	PORT_Init	Performs initialization necessary to control port functions.
	PORT_UserInit	Performs user-defined initialization relating to the port.
	PORT_ChangePmnInput	Switches the pin's I/O mode from output mode to input mode.
	PORT_ChangePmnOutput	Switches the pin's I/O mode from input mode to output mode.
Interrupt	INTP_Init	Performs initialization necessary to control the external interrupt INTP*n* functions.
	INTP_UserInit	Performs user-defined initialization relating to the external interrupt INTP*n* functions.
	KEY_Init	Performs initialization necessary to control the key interrupt INTKR functions.

Peripheral Function	API Function Name	Function
Interrupt	KEY_UserInit	Performs user-defined initialization relating to the key interrupt INTKR functions.
	INT_MaskableInterruptEnable	Disables/enables the acceptance of the maskable interrupts.
	INTPn_Disable	Disables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).
	INTPn_Enable	Enables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).
	KEY_Disable	Disables the acceptance of the key interrupts INTKR.
	KEY_Enable	Enables the acceptance of the key interrupts INTKR.
Serial	SAUm_Init	Performs initialization necessary to control the serial array unit and serial interface functions.
	SAUm_UserInit	Performs user-defined initialization related to the serial array unit and serial interface functions.
	SAUm_PowerOff	Halts the clock supplied to the serial array unit.
	UARTn_Init	Performs initialization of the serial interface (UART) channel.
	UARTn_Start	Sets UART communication to standby mode.
	UARTn_Stop	Ends UART communication.
	UARTn_SendData	Starts UART data transmission.
	UARTn_ReceiveData	Starts UART data reception.
	UARTn_SendEndCallback	Performs processing in response to the UART transmission complete interrupt INTST*n*.
	UARTn_ReceiveEndCallback	Performs processing in response to the UART reception complete interrupt INTSR*n*.
	UARTn_SoftOverRunCallback	Performs processing in response to the UART reception complete interrupt INTSR*n*.
	UARTn_ErrorCallback	Performs processing in response to the UART communication error interrupt INTSRE*n*.
	CSImn_Init	Performs initialization of the serial interface (CSI) channel.
	CSImn_Start	Sets CSI communication to standby mode.
	CSImn_Stop	Ends CSI communication.
	CSImn_SendData	Starts CSI data transmission.
	CSImn_ReceiveData	Starts CSI data reception.
	CSImn_SendReceiveData	Starts CSI data transmission/reception.
	CSImn_SendEndCallback	Performs processing in response to the CSI communication complete interrupt INTCSI*mn*.
	CSImn_ReceiveEndCallback	Performs processing in response to the CSI communication complete interrupt INTCSI*mn*.
	CSImn_ErrorCallback	Performs processing in response to the CSI communication error interrupt INTSRE*n*.
	IICmn_Init	Performs initialization of the serial interface (simple IIC) channel.
	IICmn_Stop	Ends simple IIC communication.

Peripheral Function	API Function Name	Function
Serial	IICmn_MasterSendStart	Starts simple IIC master transmission.
	IICmn_MasterReceiveStart	Starts simple IIC master reception.
	IICmn_StartCondition	Generates start conditions.
	IICmn_StopCondition	Generates stop conditions.
	IICmn_MasterSendEndCallback	Performs processing in response to the IIC*mn* communication complete interrupt INTIIC*mn*.
	IICmn_MasterReceiveEndCallback	Performs processing in response to the IIC*mn* communication complete interrupt INTIIC*mn*.
	IICmn_MasterErrorCallback	Performs processing in response to detection of parity error (ACK error) in simple IIC communication.
	UARTFn_Init	Performs initialization of the serial interface (UARTF*n*).
	UARTFn_PowerOff	Halts the clock supplied to the serial interface (UARTF*n*).
	UARTFn_Start	Sets UARTF communication to standby mode.
	UARTFn_Stop	Ends UARTF communication.
	UARTFn_SendData	Starts UARTF data transmission.
	UARTFn_ReceiveData	Starts UARTF data reception.
	UARTFn_SetComparisonData	Sets the data to compare to the received data.
	UARTFn_DataComparisonEnable	Starts the data comparison.
	UARTFn_DataComparisonDisable	Ends the data comparison.
	UARTFn_SendEndCallback	Performs processing in response to the transmission interrupt INTLT*n*.
	UARTFn_ReceiveEndCallback	Performs processing in response to the reception complete interrupt INTLR*n*.
	UARTFn_SoftOverRunCallback	Performs processing in response to the reception complete interrupt INTLR*n*.
	UARTFn_ExpBitCetectCallback	Performs processing in response to the status interrupt INTLS*n*.
	UARTFn_IDMatchCallback	Performs processing in response to the status interrupt INTLS*n*.
	UARTFn_ErrorCallback	Performs processing in response to the status interrupt INTLS*n*.
	IICA_Init	Performs initialization of the serial interface (IICA).
	IICA_UserInit	Performs user-defined initialization of the serial interface (IICA).
	IICA_PowerOff	Halts the clock supplied to the serial interface (IICA).
	IICA_Stop	Ends IICA communication.
	IICA_MasterSendStart	Starts IICA master transmission.
	IICA_MasterReceiveStart	Starts IICA master reception.
	IICA_StopCondition	Generates stop conditions.
	IICA_MasterSendEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA.

Peripheral Function	API Function Name	Function
Serial	IICA_MasterReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA.
	IICA_MasterErrorCallback	Performs processing in response to detection of error in IICA master communication.
	IICA_SlaveSendStart	Starts IICA slave transmission.
	IICA_SlaveReceiveStart	Starts IICA slave reception.
	IICA_SlaveSendEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA.
	IICA_SlaveReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA.
	IICA_SlaveErrorCallback	Performs processing in response to detection of error in IICA slave communication.
	IICA_GetStopConditionCallback	Performs processing in response to detection of stop condition in IICA slave communication.
	IICn_Init	Performs initialization of the serial interface (IIC*n*).
	IICn_UserInit	Performs user-defined initialization of the serial interface (IIC*n*).
	IICn_Stop	Ends IIC*n* communication.
	IICn_MasterSendStart	Starts IIC*n* master transmission.
	IICn_MasterReceiveStart	Starts IIC*n* master reception.
	IICn_MasterSendEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
	IICn_MasterReceiveEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
	IICn_MasterErrorCallback	Performs processing in response to detection of error in IIC*n* master communication.
	IICn_SlaveSendStart	Starts IIC*n* slave transmission.
	IICn_SlaveReceiveStart	Starts IIC*n* slave reception.
	IICn_SlaveSendEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
	IICn_SlaveReceiveEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
	IICn_SlaveErrorCallback	Performs processing in response to detection of error in IIC*n* slave communication.
	IICn_GetStopConditionCallback	Performs processing in response to detection of stop condition in IIC*n* slave communication.
Operational Amplifier	OPAMP_Init	Performs initialization necessary to control operational amplifier functions.
	OPAMP_UserInit	Performs user-defined initialization relating to the operational amplifier.
	AMPn_Start	Starts the operation of opeational amplifier *n* (single AMP mode).

Peripheral Function	API Function Name	Function
Operational Amplifier	AMPn_Stop	Ends the operation of operational amplifier *n* (single AMP mode).
Comparator/PGA	CMPPGA_Init	Performs initialization necessary to control comparator/programmable gain amplifiers functions.
	CMPPGA_UserInit	Performs user-defined initialization relating to the comparator/programmable gain amplifiers.
	CMPPGA_PowerOff	Halts the clock supplied to the comparator/programmable gain amplifiers.
	CMPPGA_Start	Starts the operation of comparator/programmable gain amplifier.
	CMPPGA_Stop	Ends the operation of comparator/programmable gain amplifier.
	CMPPGA_ChangeCMPnRefVoltage	Sets comparator *n* internal reference voltage.
	CMPPGA_ChangePGAFactor	Sets the input voltage amplification factor of a programmable gain amplifier.
A/D Converter	AD_Init	Performs initialization necessary to control A/D converter functions.
	AD_UserInit	Performs user-defined initialization relating to the A/D converter.
	AD_PowerOff	Halts the clock supplied to the A/D converter.
	AD_ComparatorOn	Enables operation of voltage converter.
	AD_ComparatorOff	Disables operation of voltage converter.
	AD_Start	Starts A/D conversion.
	AD_Stop	Ends A/D conversion.
	AD_SelectADChannel	Configures the analog voltage input pin for A/D conversion.
	AD_Read	Reads the results of A/D conversion.
	AD_ReadByte	Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).
D/A Converter	DA_Init	Performs initialization necessary to control D/A converter functions.
	DA_UserInit	Performs user-defined initialization relating to the D/A converter.
	DA_PowerOff	Halts the clock supplied to the D/A converter.
	DAn_Start	Starts D/A conversion.
	DAn_Stop	Ends D/A conversion.
	DAn_SetValue	Sets the initial analog voltage output to the ANO*n* pin.
	DAn_Set8BitsValue	Sets the initial analog voltage (8 bits) output to the ANO*n* pin.
	DAn_Set12BitsValue	Sets the initial analog voltage (12 bits) output to the ANO*n* pin.
Timer	TAUm_Init	Performs initialization necessary to control timer array unit functions.

Peripheral Function	API Function Name	Function
Timer	TAUm_UserInit	Performs user-defined initialization relating to the timer array unit.
	TAUm_PowerOff	Halts the clock supplied to the timer array unit.
	TAUm_Channeln_Start	Starts the count for channel *n*.
	TAUm_Channeln_Stop	Ends the count for channel *n*.
	TAUm_Channeln_ChangeCondition	Changes the counter value.
	TAUm_Channeln_ChangeTimerCondition	Changes the counter value.
	TAUm_Channeln_GetPulseWidth	Captures the high/low-level width measured between pulses of the signal (pulses) input to the TI*mn* pin.
	TAUm_Channeln_ChangeDuty	Changes the duty ratio of the PWM signal output to the TO*mn* pin.
	TAUm_Channeln_SoftWareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.
Watchdog Timer	WDT_Init	Performs initialization necessary to control watchdog timer functions.
	WDT_UserInit	Performs user-defined initialization relating to the watchdog timer.
	WDT_Restart	Clears the watchdog timer counter and resumes counting.
Real-time Clock	RTC_Init	Performs initialization necessary to control real-time counter functions.
	RTC_UserInit	Performs user-defined initialization relating to the real-time counter.
	RTC_PowerOff	Halts the clock supplied to the real-time counter.
	RTC_CounterEnable	Starts the count of the real-time counter (year, month, week-day, day, hour, minute, second).
	RTC_CounterDisable	Ends the count of the real-time counter (year, month, weekday, day, hour, minute, second).
	RTC_SetHourSystem	Sets the clock type (12-hour or 24-hour clock) of the real-time counter.
	RTC_CounterSet	Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
	RTC_CounterGet	Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
	RTC_ConstPeriodInterruptEnable	Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.
	RTC_ConstPeriodInterruptDisable	Ends the cyclic interrupt function.
	RTC_ConstPeriodInterruptCallback	Performs processing in response to the cyclic interrupt INTRTC.
	RTC_AlarmEnable	Starts the alarm interrupt function.
	RTC_AlarmDisable	Ends the alarm interrupt function.
	RTC_AlarmSet	Sets the alarm conditions (weekday, hour, minute).
	RTC_AlarmGet	Reads the alarm conditions (weekday, hour, minute).

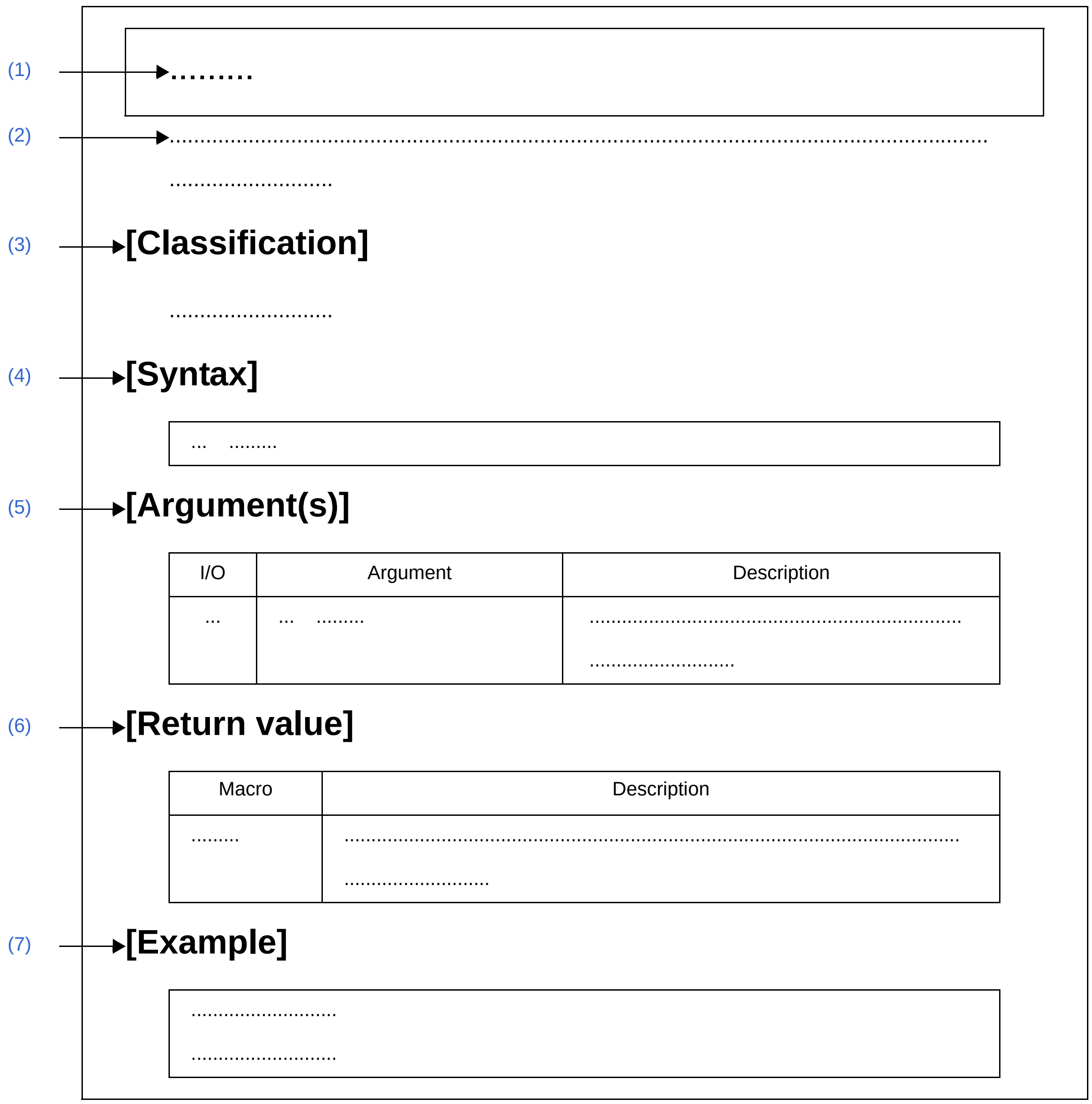
Peripheral Function	API Function Name	Function
Real-time Clock	RTC_AlarmInterruptCallback	Performs processing in response to the alarm interrupt INTRTC.
	RTC_IntervalStart	Starts the interval interrupt function.
	RTC_IntervalStop	Ends the interval interrupt function.
	RTC_IntervalInterruptEnable	Sets the cycle of the interrupts INTRTCI, then starts the interval interrupt function.
	RTC_IntervalInterruptDisable	Ends the interval interrupt function.
	RTC_RTC1HZ_OutputEnable	Enables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
	RTC_RTC1HZ_OutputDisable	Disables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
	RTC_RTCCL_OutputEnable	Enables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
	RTC_RTCCL_OutputDisable	Disables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
	RTC_RTCDIV_OutputEnable	Enables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
	RTC_RTCDIV_OutputDisable	Disables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
	RTC_ChangeCorrectionValue	Changes the timing and correction value for correcting clock errors.
Clock Output	PCL_Init	Performs initialization necessary to control clock output control circuit functions.
	PCL_UserInit	Performs user-defined initialization relating to the clock output control circuits.
	PCL_Start	Starts clock output.
	PCL_Stop	Ends clock output.
	PCL_ChangeFreq	Changes the output clock to the PCL pin.
Clock Output/Buzzer Output	PCLBUZn_Init	Performs initialization necessary to control clock/buzzer output control circuit functions.
	PCLBUZn_UserInit	Performs user-defined initialization relating to the clock/buzzer output control circuits.
	PCLBUZn_Start	Starts clock/buzzer output.
	PCLBUZn_Stop	Ends clock/buzzer output.
	PCLBUZn_ChangeFreq	Changes the output clock to the PCLBUZ*n* pin.
LCD Controller/Driver	LCD_Init	Performs initialization necessary to control LCD controller/driver functions.
	LCD_UserInit	Performs user-defined initialization relating to the LCD controller/driver.
	LCD_DisplayOn	Sets the LCD controller/driver to "display on" status.
	LCD_DisplayOff	Sets the LCD controller/driver to "display off" status.

Peripheral Function	API Function Name	Function
LCD Controller/Driver	LCD_VoltageOn	Enables operation of the LCD controller/driver's voltage boost circuit and capacitor split circuit, then outputs the deselect signal from the segment pin.
	LCD_VoltageOff	Halts operation of the LCD controller/driver's voltage boost circuit and capacitor split circuit, then outputs the groundlevel signal from the segment/common pin.
DMA	DMAn_Init	Performs initialization necessary to control DMA controller functions.
	DMAn_UserInit	Performs user-defined initialization relating to the DMA controller.
	DMAn_Enable	Enables operation of channel *n*.
	DMAn_Disable	Disables operation of channel *n*.
	DMAn_Hold	Holds a DMA start request.
	DMAn_Restart	Releases hold on a DMA start request.
	DMAn_CheckStatus	Reads the transfer status (transfer complete/transfer ongoing).
	DMAn_SetData	Sets the RAM address of the transfer source/destination, and the number of times the data has been transferred.
	DMAn_SoftwareTriggerOn	Starts DMA transfer when DMA operation is enabled.
LVI	LVI_Init	Performs initialization necessary to control low-voltage detector functions.
	LVI_UserInit	Performs user-defined initialization relating to the low-voltage detector.
	LVI_InterruptModeStart	Starts low-voltage detection (when in interrupt generation mode).
	LVI_ResetModeStart	Starts low-voltage detection (when in internal reset mode).
	LVI_Stop	Stops low-voltage detection.
	LVI_SetLVILevel	Sets the low-voltage detection level.

C.3 Function Reference

This section describes the API functions output by 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 System

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

Table C-2. API Functions: [System]

API Function Name	Function
CLOCK_Init	Performs initialization required to control the clock generator, on-chip debug, and etc. .
CLOCK_UserInit	Performs user-defined initialization relating to the clock generator, on-chip debug, and etc. .
CG_ReadResetSource	Performs processing in response to RESET signal.
CG_ChangeClockMode	Changes the CPU clock/peripheral hardware clock.
CG_ChangeFrequency	Changes the division ratio of the CPU clock/peripheral hardware clock.
CG_SelectPowerSaveMode	Configures the CPU's standby function.
CG_SelectStabTime	Configures the oscillation stabilization time of the X1 clock.

CLOCK_Init

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

[Classification]

CG_system.c

[Syntax]

```
void    CLOCK_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

CLOCK_UserInit

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

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

[Classification]

CG_system_user.c

[Syntax]

```
void    CLOCK_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

CG_ReadResetSource

Performs processing in response to RESET signal.

[Classification]

CG_system_user.c

[Syntax]

```
void    CG_ReadResetSource ( void );
```

[Argument(s)]

None.

[Return value]

None.

[Example]

Below are examples of the different processes executing depending on the RESET signal trigger.

[CG_Systeminit.c]

```
void systeminit ( void ) {
    CG_ReadResetSource ();      /* Processes executed by RESET signal trigger */
    ......
}
```

[CG_system_user.c]

```
#include    "CG_macrodriver.h"
void CG_ReadResetSource ( void ) {
    UCHAR   flag = RESF;        /* Reset control flag register: Obtain RESF contents */
    if ( flag & 0x1 ) {         /* Trigger identification: Check LVIRF flag */
        ......  /* Internal reset request by low-voltage detector */
    } else if ( flag & 0x10 ) { /* Trigger identification: Check WDRF flag */
        ......  /* Internal reset request by watchdog timer */
    } else if ( flag & 0x80 ) { /* Trigger identification: Check TRAP flag */
        ......  /* Internal reset request by execution of illegal instruction */
    }
    ......
}
```

CG_ChangeClockMode

Changes the CPU clock/peripheral hardware clock.

[Classification]

CG_system.c

[Syntax]

```
#include     "CG_macrodriver.h"
#include     "CG_system.h"
MD_STATUS    CG_ChangeClockMode ( enum ClockMode mode );
```

[Argument(s)]

I/O	Argument	Description
I	`enum ClockMode mode;`	Clock generator type [Fx3] HIOCLK: Internal high-speed oscillation clock SYSX1CLK: X1 clock SYSEXTCLK: External main system clock FILCLK: Internal low-speed oscillation clock [Ix3] HIOCLK: Internal high-speed oscillation clock HIO40CLK: 40 MHz internal high-speed oscillation clock SYSX1CLK: X1 clock SYSEXTCLK: External main system clock SUBCLK: Subsystem clock [Kx3] HIOCLK: Internal high-speed oscillation clock SYSX1CLK: X1 clock SYSEXTCLK: External main system clock SUBCLK: Subsystem clock [Kx3-A] [Kx3-L] [Lx3] HIOCLK: Internal high-speed oscillation clock HIO20CLK: 20 MHz internal high-speed oscillation clock SYSX1CLK: X1 clock SYSEXTCLK: External main system clock SUBCLK: Subsystem clock

[Return value]

Macro	Description
MD_OK	Normal completion

Macro	Description
MD_ERROR1	Exit with error (abend) [Fx3] [Kx3] - Cannot change to the X1 clock. Exit with error (abend) [Ix3] - Cannot change to the 40 MHz internal high-speed oscillation clock. Exit with error (abend) [Kx3-A] [Kx3-L] [Lx3] - Cannot change to the 20 MHz internal high-speed oscillation clock.
MD_ERROR2	Exit with error (abend) [Fx3] [Kx3] - Cannot change to the external main system clock. Exit with error (abend) [Ix3] [Kx3-A] [Kx3-L] [Lx3] - Cannot change to the X1 clock.
MD_ERROR3	Exit with error (abend) [Fx3] - Cannot change to the internal low-speed oscillation clock. Exit with error (abend) [Ix3] [Kx3-A] [Kx3-L] [Lx3] - Cannot change to the external main system clock. Exit with error (abend) [Kx3] - Cannot change to the subsystem clock because the XT1 and XT2 pins are in input mode.
MD_ERROR4	Exit with error (abend) [Ix3] [Kx3-A] [Kx3-L] [Lx3] - Cannot change to the subsystem clock.
MD_ARGERROR	Invalid argument specification

CG_ChangeFrequency

Changes the division ratio of the CPU clock/peripheral hardware clock.

[Classification]

CG_system.c

[Syntax]

```
#include     "CG_macrodriver.h"
#include     "CG_system.h"
MD_STATUS    CG_ChangeFrequency ( enum CPUClock clock );
```

[Argument(s)]

I/O	Argument	Description
I	enum CPUClock *clock*;	Division ratio type [Fx3] SYSTEMCLOCK: fPLL SYSONEHALF: fPLL/2 SYSONEFOURTH: fPLL/4 SYSONEEIGHTH: fPLL/8 SYSONESIXTEENTH: fPLL/16 SYSONETHIRTYSECOND: fPLL/32 [Ix3] [Kx3] [Kx3-L] SYSTEMCLOCK: fMAIN SYSONEHALF: fMAIN/2 SYSONEFOURTH: fMAIN/4 SYSONEEIGHTH: fMAIN/8 SYSONESIXTEENTH: fMAIN/16 SYSONETHIRTYSECOND: fMAIN/32 [Kx3-A] [Lx3] SYSTEMCLOCK: fMAIN SYSONEHALF: fMAIN/2 SYSONEFOURTH: fMAIN/4 SYSONEEIGHTH: fMAIN/8 SYSONESIXTEENTH: fMAIN/16 SYSONETHIRTYSECOND: fMAIN/32 SUB: fSUB SUBONEHALF: fSUB/2

Remark "fPLL" signifies the frequency of the PLL clock, "fMAIN" signifies the frequency of the main system clock, and "fSUB" signifies the frequency of the subsystem clock.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CG_SelectPowerSaveMode

Configures the CPU's standby function.

[Classification]

CG_system.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_SelectPowerSaveMode ( enum PSLevel level );
```

[Argument(s)]

I/O	Argument	Description
I	`enum PSLevel level;`	Standby function type PSSTOP: STOP mode PSHALT: HALT mode

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend) - If the CPU is operating by a subsystem clock (XT1 oscillator), then STOP mode cannot be specified.
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of changing the standby function to "STOP mode".

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
void main ( void ) {
    MD_STATUS   ret;
    ......
    TAU0_PowerOff ();                           /* Stop clock supply */
    ret = CG_SelectPowerSaveMode ( PSSTOP );    /* Change to STOP mode */
    if ( ret != MD_OK ) {
        while ( 1 );
    }
    TAU0_Init ();                               /* Initialize timer array unit */
```

```
    TAU0_Channel0_Start ();                     /* Starts the channel 0 count */
    ......
}
```

CG_SelectStabTime

Configures the oscillation stabilization time of the X1 clock.

[Classification]

CG_system.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_SelectStabTime ( enum StabTime waittime );
```

[Argument(s)]

I/O	Argument	Description
I	enum StabTime *waittime*;	Oscillation stabilization time type STLEVEL0: 2^8/fx STLEVEL1: 2^9/fx STLEVEL2: 2^10/fx STLEVEL3: 2^11/fx STLEVEL4: 2^13/fx STLEVEL5: 2^15/fx STLEVEL6: 2^17/fx STLEVEL7: 2^18/fx

Remark "fx" signifies the frequency of the X1 clock.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

C.3.2 External Bus

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

Table C-3. API Functions: [External Bus]

API Function Name	Function
BUS_Init	Performs initialization necessary to control external bus interface functions (functions to connect an external bus to areas other than onboard ROM, ROM and RAM).
BUS_UserInit	Performs user-defined initialization relating to the external bus interface.
BUS_PowerOff	Halts the clock supplied to the external bus interface.

BUS_Init

Performs initialization necessary to control external bus interface functions (functions to connect an external bus to areas other than onboard ROM, ROM and RAM).

[Classification]

CG_bus.c

[Syntax]

```
void    BUS_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

BUS_UserInit

Performs user-defined initialization relating to the external bus interface.

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

[Classification]

CG_bus_user.c

[Syntax]

```
void    BUS_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

BUS_PowerOff

Halts the clock supplied to the external bus interface.

Remark Calling this API function changes the external bus interface to reset status. For this reason, writes to the control registers (memory extension mode control register: MEM) after this API function is called are ignored.

[Classification]

CG_bus.c

[Syntax]

```
void    BUS_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.3 Port

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

Table C-4. API Functions: [Port]

API Function Name	Function
PORT_Init	Performs initialization necessary to control port functions.
PORT_UserInit	Performs user-defined initialization relating to the port.
PORT_ChangePmnInput	Switches the pin's I/O mode from output mode to input mode.
PORT_ChangePmnOutput	Switches the pin's I/O mode from input mode to output mode.

PORT_Init

Performs initialization necessary to control port functions.

[Classification]

CG_port.c

[Syntax]

```
void    PORT_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

PORT_UserInit

Performs user-defined initialization relating to the port.

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

[Classification]

CG_port_user.c

[Syntax]

```
void    PORT_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

PORT_ChangeP*mn*Input

Switches the pin's I/O mode from output mode to input mode.

[Classification]

CG_port.c

[Syntax]

The format for specifying this API function differs according to whether the target pin has built-in pull-up resistance/a TLL input buffer.

- Built-in pull-up resistance: none; TLL input buffer: none

```
void    PORT_ChangePmnInput ( void );
```

- Built-in pull-up resistance: yes; TLL input buffer: none

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnInput ( BOOL enablepu );
```

- Built-in pull-up resistance: yes; TLL input buffer: yes

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnInput ( BOOL enablepu, BOOL enablettl );
```

Remark *mn* is the port number.

[Argument(s)]

I/O	Argument	Description
I	BOOL *enablepu;*	Built-in pull-up resistance used MD_TRUE: Yes MD_FALSE: No
I	BOOL *enablettl;*	Input buffer type MD_TRUE: TTL input buffer MD_FALSE: Normal input buffer

[Return value]

None.

[Example 1]

Below is shown an example where pin P00 (built-in pull-up resistance: yes; TLL input buffer: none) is changed as follows:

I/O mode type: Input mode
Built-in pull-up resistance used: Yes

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP00Input ( MD_TRUE );    /* Switch I/O mode */
    ......
}
```

[Example 2]

Below is shown an example where pin P00 (built-in pull-up resistance: yes; TLL input buffer: none) is changed as follows:

I/O mode type:	Input mode
Built-in pull-up resistance used:	No

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP00Input ( MD_FALSE );   /* Switch I/O mode */
    ......
}
```

[Example 3]

Below is shown an example where pin P04 (built-in pull-up resistance: yes; TLL input buffer: yes) is changed as follows:

I/O mode type:	Input mode
Built-in pull-up resistance used:	No
Input buffer type:	TTL input buffer

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP04Input ( MD_FALSE, MD_TRUE );  /* Switch I/O mode */
    ......
}
```

PORT_ChangeP*mn*Output

Switches the pin's I/O mode from input mode to output mode.

[Classification]

CG_port.c

[Syntax]

If the target device is a 78K0R/Fx3, then the format for specifying this API function differs depending on whether N-ch open-drain output is being performed via the target pin, and whether slow mode is specified.

- N-ch open drain output: none; Slow mode: none [Fx3]

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnOutput ( BOOL initialvalue );
```

- N-ch open drain output: yes; Slow mode: none [Fx3]

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnOutput ( BOOL enablench, BOOL initialvalue );
```

- N-ch open drain output: none; Slow mode: yes [Fx3]

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnOutput ( BOOL enableslow, BOOL initialvalue );
```

- N-ch open drain output: yes; Slow mode: yes [Fx3]

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnOutput ( BOOL enablench, BOOL enableslow, BOOL initialvalue );
```

If the target device is 78K0R/Ix3, 78K0R/Kx3, 78K0R/Kx3-A, 78K0R/Kx3-L or 78K0R/Lx3, then the format for specifying this API function differs according to whether the target pin conducts N-ch open drain output.

- N-ch open drain output: none [Ix3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnOutput ( BOOL initialvalue );
```

- N-ch open drain output: yes [Ix3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnOutput ( BOOL enablench, BOOL initialvalue );
```

Remark *nm* is the port number.

[Argument(s)]

I/O	Argument	Description
I	BOOL *enablench*;	Output mode type MD_TRUE: N-ch open drain output (V_{DD} withstand voltage) mode MD_FALSE: Normal output mode
I	BOOL *enableslow*;	Output mode type MD_TRUE: Slow mode MD_FALSE: Normal mode
I	BOOL *initialvalue*;	Initial output value MD_SET: Output HIGH level "1" MD_CLEAR: Output LOW level "0"

[Return value]

None.

[Example 1]

Below is shown an example where pin P00 (N-ch open drain output: none) is changed as follows:

I/O mode type: Output mode
Initial output value: Output HIGH level "1"

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP00Output ( MD_SET );    /* Switch I/O mode */
    ......
}
```

[Example 2]

Below is shown an example where pin P04 (N-ch open drain output: yes) is changed as follows:

I/O mode type: Output mode
Output mode type: N-ch open drain output (V_{DD} withstand voltage) mode
Initial output value: Output LOW level "0"

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP04Output ( MD_TRUE, MD_CLEAR ); /* Switch I/O mode */
    ......
}
```

C.3.4 Interrupt

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

Table C-5. API Functions: [Interrupt]

API Function Name	Function
INTP_Init	Performs initialization necessary to control the external interrupt INTP*n* functions.
INTP_UserInit	Performs user-defined initialization relating to the external interrupt INTP*n* functions.
KEY_Init	Performs initialization necessary to control the key interrupt INTKR functions.
KEY_UserInit	Performs user-defined initialization relating to the key interrupt INTKR functions.
INT_MaskableInterruptEnable	Disables/enables the acceptance of the maskable interrupts.
INTPn_Disable	Disables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).
INTPn_Enable	Enables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).
KEY_Disable	Disables the acceptance of the key interrupts INTKR.
KEY_Enable	Enables the acceptance of the key interrupts INTKR.

INTP_Init

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

[Classification]

CG_int.c

[Syntax]

```
void    INTP_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

INTP_UserInit

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

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

[Classification]

CG_int_user.c

[Syntax]

```
void    INTP_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

KEY_Init

Performs initialization necessary to control the key interrupt INTKR functions.

[Classification]

CG_int.c

[Syntax]

```
void    KEY_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

KEY_UserInit

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

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

[Classification]

CG_int_user.c

[Syntax]

```
void    KEY_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

INT_MaskableInterruptEnable

Disables/enables the acceptance of the maskable interrupts.

[Classification]

CG_int.c

[Syntax]

- [Fx3] [Ix3] [Kx3-A] [Kx3-L] [Lx3]

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
MD_STATUS   INT_MaskableInterruptEnable ( enum MaskableSource name, BOOL enableflag );
```

- [Kx3]

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void    INT_MaskableInterruptEnable ( enum MaskableSource name, BOOL enableflag );
```

[Argument(s)]

I/O	Argument	Description
I	`enum MaskableSource` *name*;	Maskable interrupt type INT_*xxx*: Maskable interrupt
I	`BOOL` *enableflag*;	Acceptance enabled/disabled MD_TRUE: Acceptance enabled MD_FALSE: Acceptance disabled

Remark See the header file CG_int.h for details about the maskable interrupt type INT_*xxx*.

[Return value]

- [Fx3] [Ix3] [Kx3-A] [Kx3-L] [Lx3]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

- [Kx3]
  None.

[Example 1]

Below is an example of disabling acceptance of the maskable interrupt INTP0.

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void main ( void ) {
    ......
    INT_MaskableInterruptEnable ( INT_INTP0, MD_FALSE );  /* Disable acceptance of maskable
interrupt INTP0 */
    ......
}
```

[Example 2]

Below is an example of enabling acceptance of the maskable interrupt INTP0.

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void main ( void ) {
    ......
    INT_MaskableInterruptEnable ( INT_INTP0, MD_TRUE );   /* Enable acceptance of maskable
interrupt INTP0 */
    ......
}
```

INTP*n*_Disable

Disables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).

[Classification]

CG_int.c

[Syntax]

```
void    INTPn_Disable ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

INTP*n*_Enable

Enables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).

[Classification]

CG_int.c

[Syntax]

```
void    INTPn_Enable ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

KEY_Disable

Disables the acceptance of the key interrupts INTKR.

[Classification]

CG_int.c

[Syntax]

```
void    KEY_Disable ( void );
```

[Argument(s)]

None.

[Return value]

None.

KEY_Enable

Enables the acceptance of the key interrupts INTKR.

[Classification]

CG_int.c

[Syntax]

```
void    KEY_Enable ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.5 Serial

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

Table C-6. API Functions: [Serial]

API Function Name	Function
SAUm_Init	Performs initialization necessary to control the serial array unit and serial interface functions.
SAUm_UserInit	Performs user-defined initialization related to the serial array unit and serial interface functions.
SAUm_PowerOff	Halts the clock supplied to the serial array unit.
UARTn_Init	Performs initialization of the serial interface (UART) channel.
UARTn_Start	Sets UART communication to standby mode.
UARTn_Stop	Ends UART communication.
UARTn_SendData	Starts UART data transmission.
UARTn_ReceiveData	Starts UART data reception.
UARTn_SendEndCallback	Performs processing in response to the UART transmission complete interrupt INTST*n*.
UARTn_ReceiveEndCallback	Performs processing in response to the UART reception complete interrupt INTSR*n*.
UARTn_SoftOverRunCallback	Performs processing in response to the serial transfer end interrupt INTSR*n*.
UARTn_ErrorCallback	Performs processing in response to the UART communication error interrupt INTSRE*n*.
CSImn_Init	Performs initialization of the serial interface (CSI) channel.
CSImn_Start	Sets CSI communication to standby mode.
CSImn_Stop	Ends CSI communication.
CSImn_SendData	Starts CSI data transmission.
CSImn_ReceiveData	Starts CSI data reception.
CSImn_SendReceiveData	Starts CSI data transmission/reception.
CSImn_SendEndCallback	Performs processing in response to the CSI communication complete interrupt INTCSI*mn*.
CSImn_ReceiveEndCallback	Performs processing in response to the CSI communication complete interrupt INTCSI*mn*.
CSImn_ErrorCallback	Performs processing in response to the CSI communication error interrupt INTSRE*n*.
IICmn_Init	Performs initialization of the serial interface (simple IIC) channel.
IICmn_Stop	Ends simple IIC communication.
IICmn_MasterSendStart	Starts simple IIC master transmission.
IICmn_MasterReceiveStart	Starts simple IIC master reception.
IICmn_StartCondition	Generates start conditions.
IICmn_StopCondition	Generates stop conditions.
IICmn_MasterSendEndCallback	Performs processing in response to the simple IIC*mn* communication complete interrupt INTIIC*mn*.
IICmn_MasterReceiveEndCallback	Performs processing in response to the simple IIC*mn* communication complete interrupt INTIIC*mn*.

API Function Name	Function
IICmn_MasterErrorCallback	Performs processing in response to detection of parity error (ACK error) in simple IIC communication.
UARTFn_Init	Performs initialization of the serial interface (UARTF*n*).
UARTFn_PowerOff	Halts the clock supplied to the serial interface (UARTF*n*).
UARTFn_Start	Sets UARTF communication to standby mode.
UARTFn_Stop	Ends UARTF communication.
UARTFn_SendData	Starts UARTF data transmission.
UARTFn_ReceiveData	Starts UARTF data reception.
UARTFn_SetComparisonData	Sets the data to compare to the received data.
UARTFn_DataComparisonEnable	Starts the data comparison.
UARTFn_DataComparisonDisable	Ends the data comparison.
UARTFn_SendEndCallback	Performs processing in response to the transmission interrupt INTLT*n*.
UARTFn_ReceiveEndCallback	Performs processing in response to the reception complete interrupt INTLR*n*.
UARTFn_SoftOverRunCallback	Performs processing in response to the reception complete interrupt INTLR*n*.
UARTFn_ExpBitCetectCallback	Performs processing in response to the status interrupt INTLS*n*.
UARTFn_IDMatchCallback	Performs processing in response to the status interrupt INTLS*n*.
UARTFn_ErrorCallback	Performs processing in response to the status interrupt INTLS*n*.
IICA_Init	Performs initialization of the serial interface (IICA).
IICA_UserInit	Performs user-defined initialization of the serial interface (IICA).
IICA_PowerOff	Halts the clock supplied to the serial interface (IICA).
IICA_Stop	Ends IICA communication.
IICA_MasterSendStart	Starts IICA master transmission.
IICA_MasterReceiveStart	Starts IICA master reception.
IICA_StopCondition	Generates stop conditions.
IICA_MasterSendEndCallback	Performs processing in response to the IICA communication complete interrupt INTI-ICA.
IICA_MasterReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTI-ICA.
IICA_MasterErrorCallback	Performs processing in response to detection of error in IICA master communication.
IICA_SlaveSendStart	Starts IICA slave transmission.
IICA_SlaveReceiveStart	Starts IICA slave reception.
IICA_SlaveSendEndCallback	Performs processing in response to the IICA communication complete interrupt INTI-ICA.
IICA_SlaveReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTI-ICA.
IICA_SlaveErrorCallback	Performs processing in response to detection of error in IICA slave communication.
IICA_GetStopConditionCallback	Performs processing in response to detection of stop condition in IICA slave communi-cation.
IICn_Init	Performs initialization of the serial interface (IIC*n*).

API Function Name	Function
IICn_UserInit	Performs user-defined initialization of the serial interface (IIC*n*).
IICn_Stop	Ends IIC*n* communication.
IICn_MasterSendStart	Starts IIC*n* master transmission.
IICn_MasterReceiveStart	Starts IIC*n* master reception.
IICn_MasterSendEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
IICn_MasterReceiveEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
IICn_MasterErrorCallback	Performs processing in response to detection of error in IIC*n* master communication.
IICn_SlaveSendStart	Starts IIC*n* slave transmission.
IICn_SlaveReceiveStart	Starts IIC*n* slave reception.
IICn_SlaveSendEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
IICn_SlaveReceiveEndCallback	Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.
IICn_SlaveErrorCallback	Performs processing in response to detection of error in IIC*n* slave communication.
IICn_GetStopConditionCallback	Performs processing in response to detection of stop condition in IIC*n* slave communication.

SAU*m*_Init

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

[Classification]

CG_serial.c

[Syntax]

```
void    SAUm_Init ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

SAU*m*_UserInit

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

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    SAUm_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

SAU*m*_PowerOff

Halts the clock supplied to the serial array unit.

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

[Classification]

CG_serial.c

[Syntax]

```
void    SAUm_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

UART*n*_Init

Performs initialization of the serial interface (UART) channel.

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

[Classification]

CG_serial.c

[Syntax]

```
void    UARTn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UART*n*_Start

Sets UART communication to standby mode.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UART*n*_Stop

Ends UART communication.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UART*n*_SendData

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, UARTn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   UARTn_SendData ( UCHAR *txbuf, USHORT txnum );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of sending a UART transmission of four bytes of fixed-length data from channel 0 one time.

[CG_main.c]

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                          /* Transmission complete flag */
void main ( void ) {
    UCHAR   txbuf[] = "ABCD";
    USHORT  txnum = 4;
    gFlag = 1;                          /* Initialize transmission complete flag */
    ......
    UART0_Start ();                     /* Start UART communication*/
    UART0_SendData ( &txbuf, txnum );   /* Start UART data transmission */
    while ( gFlag );                    /* Wait for txnum transmissions */
```

```
    ......
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                  /* Transmission complete flag */
__interrupt void MD_INTST0 ( void ) {   /* Interrupt processing for INTST0 */
    if ( gUart0TxCnt > 0 ) {
        ......
    } else {
        UART0_SendEndCallback ();       /* Call callback routine */
    }
}

void UART0_SendEndCallback ( void ) {   /* Callback routine for INTST0 */
    gFlag = 0;                          /* Set transmission complete flag */
}
```

UART*n*_ReceiveData

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 UARTn_Start is then called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   UARTn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of UART reception of four bytes of fixed-length data from channel 0 one time.

[CG_main.c]

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                              /* Reception complete flag */
void main ( void ) {
    UCHAR   rxbuf[10];
    USHORT  rxnum = 4;
    gFlag = 1;                              /* Initialize reception complete flag */
    ......
    UART0_ReceiveData ( &rxbuf, rxnum );    /* Start UART data reception */
    UART0_Start ();                         /* Start UART communication */
    while ( gFlag );                        /* Wait for rxnum receptions */
```

```
    ......
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* Reception complete flag */
__interrupt void MD_INTSR0 ( void ) {       /* Interrupt processing for INTSR0 */
    ......
    if ( gUart0RxLen > gUart0RxCnt ) {
        ......
        if ( gUart0RxLen == gUart0RxCnt ) {
            UART0_ReceiveEndCallback ();    /* Call callback routine */
        }
    }
}

void UART0_ReceiveEndCallback ( void ) {    /* Callback routine for INTSR0 */
    gFlag = 0;                              /* Set reception complete flag */
}
```

UART*n*_SendEndCallback

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

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTn_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UART*n*_ReceiveEndCallback

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

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTn_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UART*n*_SoftOverRunCallback

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

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

[Classification]

CG_serial_user.c

[Syntax]

- [Fx3]

```
void    UARTn_SoftOverRunCallback ( void );
```

- [Ix3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

```
#include    "CG_macrodriver.h"
void    UARTn_SoftOverRunCallback ( UCHAR rx_data );
```

Remark *n* is the channel number.

[Argument(s)]

- [Fx3]
  None.

- [Ix3] [Kx3] [Kx3-A] [Kx3-L] [Lx3]

I/O	Argument	Description
O	UCHAR *rx_data*;	Receive data (greater than the parameter *rxnum* specified for UARTn_ReceiveData)

[Return value]

None.

UART*n*_ErrorCallback

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

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

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    UARTn_ErrorCallback ( UCHAR err_type );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`UCHAR 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.

[Example]

Below are examples of callback processing by the trigger for the UART communication error interrupt.

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
__interrupt void MD_INTSRE0 ( void ) {          /* Interrupt processing for INTSRE0 */
    UCHAR   err_type;
    ......
    UART0_ErrorCallback ( err_type );           /* Call callback routine */
}

void UART0_ErrorCallback ( UCHAR err_type ) {   /* Callback routine for INTSRE0 */
    if ( err_type & 0x1 ) {                     /* Determine trigger */
        ......  /* Callback processing in response to overrun error */
    } else if ( err_type & 0x2 ) {              /* Determine trigger */
        ......  /* Callback processing in response to parity error */
    } else if ( err_type & 0x4 ) {              /* Determine trigger */
```

```
        ......  /* Callback processing in response to framing error */
    }
}
```

CSI*mn*_Init

Performs initialization of the serial interface (CSI) channel.

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

[Classification]

CG_serial.c

[Syntax]

```
void    CSImn_Init ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

CSI*mn*_Start

Sets CSI communication to standby mode.

[Classification]

CG_serial.c

[Syntax]

```
void    CSImn_Start ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

CSI*mn*_Stop

Ends CSI communication.

[Classification]

CG_serial.c

[Syntax]

```
void    CSImn_Stop ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

CSI*mn*_SendData

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, CSImn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSImn_SendData ( UCHAR *txbuf, USHORT txnum );
```

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

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of sending a CSI transmission of four bytes of fixed-length data from channel 00 one time.

[CG_main.c]

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                          /* Transmission complete flag */
void main ( void ) {
    UCHAR   txbuf[] = "ABCD";
    USHORT  txnum = 4;
    gFlag = 1;                          /* Initialize transmission complete flag */
    ......
    CSI00_Start ();                     /* Start CSI communication */
    CSI00_SendData ( &txbuf, txnum );   /* Start CSI transmission */
    while ( gFlag );                    /* Wait for txnum transmissions */
```

```
    ......
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                  /* Transmission complete flag */
__interrupt void MD_INTCSI00 ( void ) {   /* Interrupt processing for INTCSI00 */
    if ( gCsi00TxCnt > 0 ) {
        ......
    } else {
        CSI00_SendEndCallback ();         /* Call callback routine */
    }
}

void CSI00_SendEndCallback ( void ) {     /* Callback routine for INTCSI00 */
    gFlag = 0;                          /* Set transmission complete flag */
}
```

CSI*mn*_ReceiveData

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, CSImn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSImn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

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

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of receiving a CSI transmission of four bytes of fixed-length data from channel 00 one time.

[CG_main.c]

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                              /* Reception complete flag */
void main ( void ) {
    UCHAR    rxbuf[10];
    USHORT   rxnum = 4;
    gFlag = 1;                              /* Initialize reception complete flag */
    ......
    CSI00_Start ();                         /* Start CSI communication */
    CSI00_ReceiveData ( &rxbuf, rxnum );    /* Start CSI reception */
    while ( gFlag );                        /* Wait for rxnum receptions */
```

```
    ......
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* Reception complete flag */
__interrupt void MD_INTCSI00 ( void ) {     /* Interrupt processing for INTCSI00 */
    if ( gCsi00RxCnt < gCsi00RxLen ) {
        ......
        if ( gCsi00RxCnt == gCsi00RxLen ) {
            CSI00_ReceiveEndCallback ();    /* Call callback routine */
        } else {
            ......
        }
    }
}

void CSI00_ReceiveEndCallback ( void ) {    /* Callback routine for INTCSI00 */
    gFlag = 0;                              /* Set reception complete flag */
}
```

CSI*mn*_SendReceiveData

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, CSImn_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSImn_SendReceiveData ( UCHAR *txbuf, USHORT txnum, UCHAR *rxbuf );
```

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

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of sending and receiving a CSI transmission of four bytes of fixed-length data from channel 00 one time.

[CG_main.c]

```
#include    "CG_macrodriver.h"
BOOL    gSflag;                                        /* Transmission complete flag */
void main ( void ) {
    UCHAR   txbuf[] = "0123";
    USHORT  txnum = 4;
    UCHAR   rxbuf[10];
```

```
    gSflag = 1;                                         /* Initialize flag */
    ......
    CSI00_Start ();                                     /* Start CSI communication */
    CSI00_SendReceiveData ( &txbuf, txnum, &rxbuf );    /* Start CSI send/receive */
    while ( gSlag );                                /* Wait for txnum transmissions/receptions
*/
    ......
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gSflag;                         /* Transmission complete flag */
__interrupt void MD_INTCSI00 ( void ) {         /* Interrupt processing for INTCSI00 */
   if ( gCsi00TxCnt > 0 ) {
        ......
    } else {
        ......
        CSI00_SendEndCallback ();               /* Call callback routine */
    }
    ......
}

void CSI00_SendEndCallback ( void ) {           /* Callback routine for INTCSI00 */
    gSflag = 0;                                 /* Set transmission complete flag */
}
```

CSI*mn*_SendEndCallback

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

Remark This API function is called as the callback routine of interrupt process MD_INTCSI*mn* corresponding to the CSI communication complete interrupt INTCSI*mn* (performed when number of transmission data specified by CSImn_SendData parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSImn_SendEndCallback ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

CSI*mn*_ReceiveEndCallback

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

Remark This API function is called as the callback routine of interrupt process MD_INTCSI*mn* corresponding to the CSI communication complete interrupt INTCSI*mn* (performed when number of received data specified by CSImn_ReceiveData parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSImn_ReceiveEndCallback ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

CSI*mn*_ErrorCallback

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

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

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    CSImn_ErrorCallback ( UCHAR err_type );
```

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

[Argument(s)]

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

[Return value]

None.

[Example]

Below are examples of callback processing by the trigger for the CSI communication error interrupt.

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
__interrupt void MD_INTSRE0 ( void ) {          /* Interrupt processing for INTSRE0 */
    UCHAR   err_type;
    ......
    CSI00_ErrorCallback ( err_type );           /* Call callback routine */
}

void CSI00_ErrorCallback ( UCHAR err_type ) {   /* Callback routine for INTSRE0 */
    if ( err_type & 0x1 ) {                     /* Determine trigger */
        ......  /* Callback processing in response to overrun error */
    }
}
```

IIC*mn*_Init

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

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

[Classification]

CG_serial.c

[Syntax]

```
void    IICmn_Init ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

IIC*mn*_Stop

Ends simple IIC communication.

[Classification]

CG_serial.c

[Syntax]

```
void    IICmn_Stop ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

IIC*mn*_MasterSendStart

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]

CG_serial.c

[Syntax]

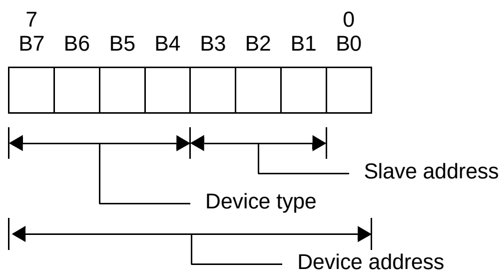
```
#include    "CG_macrodriver.h"
void    IICmn_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum );
```

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

[Argument(s)]

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

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



[Return value]

None.

IIC*mn*_MasterReceiveStart

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]

CG_serial.c

[Syntax]

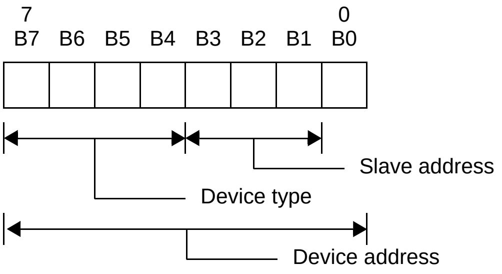
```
#include    "CG_macrodriver.h"
void    IICmn_MasterReceiveStart ( UCHAR adr, UCHAR *rxbuf, USHORT rxnum );
```

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

[Argument(s)]

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

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



[Return value]

None.

IIC*mn*_StartCondition

Generates start conditions.

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

[Classification]

CG_serial.c

[Syntax]

```
void    IICmn_StartCondition ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

IIC*mn*_StopCondition

Generates stop conditions.

[Classification]

CG_serial.c

[Syntax]

```
void    IICmn_StopCondition ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

IIC*mn*_MasterSendEndCallback

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

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICmn_MasterSendEndCallback ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

IIC*mn*_MasterReceiveEndCallback

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

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*mn* corresponding to the simple IIC*mn* communication complete interrupt INTIIC*mn* (performed when number of received data specified by IICmn_MasterReceiveStart parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICmn_MasterReceiveEndCallback ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

IIC*mn*_MasterErrorCallback

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

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    IICmn_MasterErrorCallback ( 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.

UARTF*n*_Init

Performs initialization of the serial interface (UARTF*n*).

[Classification]

CG_serial.c

[Syntax]

```
void    UARTFn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_PowerOff

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

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

[Classification]

CG_serial.c

[Syntax]

```
void    UARTFn_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_Start

Sets UARTF communication to standby mode.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTFn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_Stop

Ends UARTF communication.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTFn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_SendData

Starts UARTF data transmission.

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

2. When performing a UARTF transmission, UARTFn_Start must be called before this API function is called.

3. If the serial interface (UARTF*n*) is used in expansion bit mode, then store the data to send in the buffer specified by parameter *txbuf*, in the following format.

"8-bit data", "Expansion bit", "8-bit data", "Expansion bit", ...

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   UARTFn_SendData ( UCHAR *txbuf, USHORT txnum );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

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

UARTF*n*_ReceiveData

Starts UARTF data reception.

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

2. Actual UARTF reception starts after this API function is called, and UARTFn_Start is then called.

3. If the serial interface (UARTF*n*) is used in expansion bit mode, then the received data is stored in the buffer specified by parameter *rxbuf*, in the following format.

"8-bit data", "Expansion bit", "8-bit data", "Expansion bit", ...

[Classification]

CG_serial.c

[Syntax]

```
#include     "CG_macrodriver.h"
MD_STATUS    UARTFn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

UARTF*n*_SetComparisonData

Sets the data to compare to the received data.

Remark The value specified in parameter *comdata* is set to LIN-UART*n* ID setting register (UF*n*ID).

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    UARTFn_SetComparisonData ( UCHAR comdata );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`UCHAR comdata;`	Data to compare

[Return value]

None.

UARTF*n*_DataComparisonEnable

Starts the data comparison.

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_DataComparisonEnable ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_DataComparisonDisable

Ends the data comparison.

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_DataComparisonDisable ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_SendEndCallback

Performs processing in response to the transmission interrupt INTLT*n*.

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_ReceiveEndCallback

Performs processing in response to the reception complete interrupt INTLR*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTLR*n* corresponding to the reception complete interrupt INTLR*n* (performed when number of received data specified by UARTFn_ReceiveData parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_SoftOverRunCallback

Performs processing in response to the reception complete interrupt INTLR*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTLR*n* corresponding to the reception complete interrupt INTLR*n* (process performed when the amount of data received is greater than the parameter *rxnum* specified for UARTFn_ReceiveData).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_SoftOverRunCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_ExpBitCetectCallback

Performs processing in response to the status interrupt INTLS*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTLS*n* corresponding to the status interrupt INTLS*n* (processing when the expansion bit is received).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_ExpBitCetectCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_IDMatchCallback

Performs processing in response to the status interrupt INTLS*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTLS*n* corresponding to the status interrupt INTLS*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTFn_IDMatchCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTF*n*_ErrorCallback

Performs processing in response to the status interrupt INTLS*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTLS*n* corresponding to the status interrupt INTLS*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    UARTFn_ErrorCallback ( UCHAR err_type );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

IICA_Init

Performs initialization of the serial interface (IICA).

[Classification]

CG_serial.c

[Syntax]

```
void    IICA_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_UserInit

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

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICA_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_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]

CG_serial.c

[Syntax]

```
void    IICA_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_Stop

Ends IICA communication.

[Classification]

CG_serial.c

[Syntax]

```
void    IICA_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_MasterSendStart

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]

CG_serial.c

[Syntax]

```
#include     "CG_macrodriver.h"
MD_STATUS    IICA_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum, UCHAR wait );
```

[Argument(s)]

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

IICA_MasterReceiveStart

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]

CG_serial.c

[Syntax]

```
#include     "CG_macrodriver.h"
MD_STATUS    IICA_MasterReceiveStart ( UCHAR adr, UCHAR *rxbuf, USHORT rxnum, UCHAR wait );
```

[Argument(s)]

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

IICA_StopCondition

Generates stop conditions.

[Classification]

CG_serial.c

[Syntax]

```
void    IICA_StopCondition ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_MasterSendEndCallback

Performs processing in response to the IICA communication complete interrupt INTIICA.

Remark This API function is called as the callback routine of interrupt process MD_INTIICA corresponding to the IICA communication complete interrupt INTIICA.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICA_MasterSendEndCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_MasterReceiveEndCallback

Performs processing in response to the IICA communication complete interrupt INTIICA.

Remark This API function is called as the callback routine of interrupt process MD_INTIICA corresponding to the IICA communication complete interrupt INTIICA.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICA_MasterReceiveEndCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_MasterErrorCallback

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

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    IICA_MasterErrorCallback ( MD_STATUS flag  );
```

[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.

IICA_SlaveSendStart

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]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    IICA_SlaveSendStart ( UCHAR *txbuf, USHORT txnum );
```

[Argument(s)]

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

[Return value]

None.

IICA_SlaveReceiveStart

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]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    IICA_SlaveReceiveStart ( UCHAR *rxbuf, USHORT rxnum );
```

[Argument(s)]

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

[Return value]

None.

IICA_SlaveSendEndCallback

Performs processing in response to the IICA communication complete interrupt INTIICA.

Remark This API function is called as the callback routine of interrupt process MD_INTIICA corresponding to the IICA communication complete interrupt INTIICA.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICA_SlaveSendEndCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_SlaveReceiveEndCallback

Performs processing in response to the IICA communication complete interrupt INTIICA.

Remark This API function is called as the callback routine of interrupt process MD_INTIICA corresponding to the IICA communication complete interrupt INTIICA.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICA_SlaveReceiveEndCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

IICA_SlaveErrorCallback

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

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    IICA_SlaveErrorCallback ( MD_STATUS flag  );
```

[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.

IICA_GetStopConditionCallback

Performs processing in response to detection of stop condition in IICA slave communication.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICA_GetStopConditionCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

IIC*n*_Init

Performs initialization of the serial interface (IIC*n*).

[Classification]

CG_serial.c

[Syntax]

```
void    IICn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_UserInit

Performs user-defined initialization of the serial interface (IIC*n*).

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

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_Stop

Ends IIC*n* communication.

[Classification]

CG_serial.c

[Syntax]

```
void    IICn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_MasterSendStart

Starts IIC*n* master transmission.

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

[Classification]

CG_serial.c

[Syntax]

```
#include     "CG_macrodriver.h"
MD_STATUS    IICn_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum, UCHAR wait );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

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

IIC*n*_MasterReceiveStart

Starts IIC*n* master reception.

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

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   IICn_MasterReceiveStart ( UCHAR adr, UCHAR *rxbuf, USHORT rxnum, UCHAR wait );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

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

IIC*n*_MasterSendEndCallback

Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*n* corresponding to the IIC*n* communication complete interrupt INTIIC*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICn_MasterSendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_MasterReceiveEndCallback

Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*n* corresponding to the IIC*n* communication complete interrupt INTIIC*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICn_MasterReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_MasterErrorCallback

Performs processing in response to detection of error in IIC*n* master communication.

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    IICn_MasterErrorCallback ( 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.

IIC*n*_SlaveSendStart

Starts IIC*n* slave transmission.

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

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    IICn_SlaveSendStart ( UCHAR *txbuf, USHORT txnum );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

IIC*n*_SlaveReceiveStart

Starts IIC*n* slave reception.

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

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    IICn_SlaveReceiveStart ( UCHAR *rxbuf, USHORT rxnum );
```

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

None.

IIC*n*_SlaveSendEndCallback

Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*n* corresponding to the IIC*n* communication complete interrupt INTIIC*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICn_SlaveSendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_SlaveReceiveEndCallback

Performs processing in response to the IIC*n* communication complete interrupt INTIIC*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTIIC*n* corresponding to the IIC*n* communication complete interrupt INTIIC*n*.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICn_SlaveReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC*n*_SlaveErrorCallback

Performs processing in response to detection of error in IIC*n* slave communication.

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    IICn_SlaveErrorCallback ( 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.

IIC*n*_GetStopConditionCallback

Performs processing in response to detection of stop condition in IIC*n* slave communication.

[Classification]

CG_serial_user.c

[Syntax]

```
void    IICn_GetStopConditionCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.3.6 Operational Amplifier

Below is a list of API functions output by Code Generator for operational amplifiers use.

Table C-7. API Functions: [Operational Amplifier]

API Function Name	Function
OPAMP_Init	Performs initialization necessary to control operational amplifier functions.
OPAMP_UserInit	Performs user-defined initialization relating to the operational amplifier.
AMPn_Start	Starts the operation of opeational amplifier *n* (single AMP mode).
AMPn_Stop	Ends the operation of operational amplifier *n* (single AMP mode).

OPAMP_Init

Performs initialization necessary to control operational amplifier functions.

[Classification]

CG_opamp.c

[Syntax]

```
void    OPAMP_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

OPAMP_UserInit

Performs user-defined initialization relating to the operational amplifier.

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

[Classification]

CG_opamp_user.c

[Syntax]

```
void    OPAMP_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

AMP*n*_Start

Starts the operation of operational amplifier *n* (single AMP mode).

[Classification]

CG_opamp.c

[Syntax]

```
void    AMPn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

AMP*n*_Stop

Ends the operation of operational amplifier *n* (single AMP mode).

[Classification]

CG_opamp.c

[Syntax]

```
void    AMPn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.3.7 Comparator/PGA

Below is a list of API functions output by Code Generator for comparator/programmable gain amplifiers use.

Table C-8. API Functions: [Comparator/PGA]

API Function Name	Function
CMPPGA_Init	Performs initialization necessary to control comparator/programmable gain amplifiers functions.
CMPPGA_UserInit	Performs user-defined initialization relating to the comparator/programmable gain amplifiers.
CMPPGA_PowerOff	Halts the clock supplied to the comparator/programmable gain amplifiers.
CMPPGA_Start	Starts the operation of comparator/programmable gain amplifier.
CMPPGA_Stop	Ends the operation of comparator/programmable gain amplifier.
CMPPGA_ChangeCMPnRefVoltage	Sets comparator *n* internal reference voltage.
CMPPGA_ChangePGAFactor	Sets the input voltage amplification factor of a programmable gain amplifier.

CMPPGA_Init

Performs initialization necessary to control comparator/programmable gain amplifiers functions.

[Classification]

CG_cmppga.c

[Syntax]

```
void    CMPPGA_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

CMPPGA_UserInit

Performs user-defined initialization relating to the comparator/programmable gain amplifiers.

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

[Classification]

CG_cmppga_user.c

[Syntax]

```
void    CMPPGA_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

CMPPGA_PowerOff

Halts the clock supplied to the comparator/programmable gain amplifiers.

Remark Calling this API function changes the comparator/programmable gain amplifiers to reset status. For this reason, writes to the control registers (e.g. programmable gain amplifier control register: OAM) after this API function is called are ignored.

[Classification]

CG_cmppga.c

[Syntax]

```
void    CMPPGA_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

CMPPGA_Start

Starts the operation of comparator/programmable gain amplifier.

[Classification]

CG_cmppga.c

[Syntax]

```
void    CMPPGA_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

CMPPGA_Stop

Ends the operation of comparator/programmable gain amplifier.

[Classification]

CG_cmppga.c

[Syntax]

```
void    CMPPGA_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

CMPPGA_ChangeCMP*n*RefVoltage

Sets comparator *n* internal reference voltage.

Remark The value specified in parameter *voltage* is set to comparator *n* internal reference voltage setting register (C*n*RVM).

[Classification]

CG_cmppga.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_cmppga.h"
MD_STATUS   CMPPGA_ChangeCMPnRefVoltage ( enum CMPRefVoltage voltage );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`enum CMPRefVoltage voltage;`	Comparator *n* internal reference voltage [*n* = 0: In the case of the channel 0] CMPREFVOL0: 2AVREF/16 CMPREFVOL1: 4AVREF/16 CMPREFVOL2: 6AVREF/16 CMPREFVOL3: 8AVREF/16 CMPREFVOL4: 10AVREF/16 CMPREFVOL5: 12AVREF/16 [*n* = 1: In the case of the channel 1] CMPREFVOL0: 3AVREF/16 CMPREFVOL1: 5AVREF/16 CMPREFVOL2: 7AVREF/16 CMPREFVOL3: 9AVREF/16 CMPREFVOL4: 11AVREF/16 CMPREFVOL5: 13AVREF/16

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CMPPGA_ChangePGAFactor

Sets the input voltage amplification factor of a programmable gain amplifier.

Remark The value specified in parameter *factor* is set to programmable gain amplifier control register (OAM).

[Classification]

CG_cmppga.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_cmppga.h"
MD_STATUS   CMPPGA_ChangePGAFactor ( enum PGAFactor factor );
```

[Argument(s)]

I/O	Argument	Description
I	`enum PGAFactor factor;`	Input voltage amplification factor PGAFACTOR0: x4 PGAFACTOR1: x6 PGAFACTOR2: x8 PGAFACTOR3: x10 PGAFACTOR4: x12

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

C.3.8 A/D Converter

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

Table C-9. API Functions: [A/D Converter]

API Function Name	Function
AD_Init	Performs initialization necessary to control A/D converter functions.
AD_UserInit	Performs user-defined initialization relating to the A/D converter.
AD_PowerOff	Halts the clock supplied to the A/D converter.
AD_ComparatorOn	Enables operation of voltage converter.
AD_ComparatorOff	Disables operation of voltage converter.
AD_Start	Starts A/D conversion.
AD_Stop	Ends A/D conversion.
AD_SelectADChannel	Configures the analog voltage input pin for A/D conversion.
AD_Read	Reads the results of A/D conversion.
AD_ReadByte	Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).

AD_Init

Performs initialization necessary to control A/D converter functions.

[Classification]

CG_ad.c

[Syntax]

```
void    AD_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

AD_UserInit

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

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

[Classification]

CG_ad_user.c

[Syntax]

```
void    AD_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

AD_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 (e.g. A/D converter mode register: ADCM) after this API function is called are ignored.

[Classification]

CG_ad.c

[Syntax]

```
void    AD_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

AD_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 AD_Start.

2. On the Code Generator panel ([A/D]), 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]

CG_ad.c

[Syntax]

```
void    AD_ComparatorOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

AD_ComparatorOff

Disables operation of voltage converter.

[Classification]

CG_ad.c

[Syntax]

```
void    AD_ComparatorOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

AD_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 AD_ComparatorOn and the call to this API function.

[Classification]

CG_ad.c

[Syntax]

```
void    AD_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

[Example 1]

Below is an example of starting analog voltage via the conversion start pin selected in the Code Generator panel ([A/D]), then converting the analog voltage from the ANI1 input pin to digital. In the example, "Stop" is selected in the [Comparator operation setting] area of the Code Generator panel ([A/D]) (when performing the call to AD_ComparatorOn).

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_ad.h"
BOOL    gFlag;                          /* A/D conversion complete flag */
void main ( void ) {
    USHORT  buffer = 0;
    int     wait = 100;
    gFlag = 1;                          /* Initialize A/D conversion complete flag */
    ......
    AD_ComparatorOn ();                 /* Move to operation enabled status */
    while ( wait );             /* Ensure stabilization time (at least 1 micro second) */
    AD_Start ();                        /* Start A/D conversion */
    while ( gFlag );                    /* Wait for INTAD */
    AD_Read ( &buffer );                /* Read results of A/D conversion */
    AD_SelectADChannel ( ADCHANNEL1 );  /* Switch input pins */
    gFlag = 1;                          /* Initialize A/D conversion complete flag */
    while ( gFlag );                    /* Wait for INTAD */
    AD_Read ( &buffer );                /* Read results of A/D conversion */
```

```
    AD_SelectADChannel ( ADCHANNEL1 );  /* Switch input pins */
    gFlag = 1;                          /* Initialize A/D conversion complete flag */
    while ( gFlag );                    /* Wait for INTAD */
    AD_Read ( &buffer );                /* Read results of A/D conversion */
    AD_Stop ();                         /* End A/D conversion */
    AD_ComparatorOff ();                /* Move to operation disabled status */
    ......
}
```

[CG_ad_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                  /* A/D conversion complete flag */
__interrupt void MD_INTAD ( void ) {    /* Interrupt processing for INTAD */
    gFlag = 0;                          /* Set A/D conversion complete flag */
}
```

[Example 2]

Below is an example of starting analog voltage via the conversion start pin selected in the Code Generator panel ([A/D]), then converting the analog voltage from the ANI1 input pin to digital. In the example, "Operation" is selected in the [Comparator operation setting] area of the Code Generator panel ([A/D]) (when not performing the call to AD_ComparatorOn).

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_ad.h"
BOOL    gFlag;                          /* A/D conversion complete flag */
void main ( void ) {
    USHORT  buffer = 0;
    gFlag = 1;                          /* Initialize A/D conversion complete flag */
    ......
    AD_Start ();                        /* Start A/D conversion */
    while ( gFlag );                    /* Wait for INTAD */
    AD_Read ( &buffer );                /* Read results of A/D conversion */
    AD_SelectADChannel ( ADCHANNEL1 );  /* Switch input pins */
    gFlag = 1;                          /* Initialize A/D conversion complete flag */
    while ( gFlag );                    /* Wait for INTAD */
    AD_Read ( &buffer );                /* Read results of A/D conversion */
    AD_Stop ();                         /* End A/D conversion */
    ......
}
```

[CG_ad_user.c]

```
#include    "CG_macrodriver.h"
```

```
extern  BOOL    gFlag;                  /* A/D conversion complete flag */
__interrupt void MD_INTAD ( void ) {    /* Interrupt processing for INTAD */
    gFlag = 0;                          /* Set A/D conversion complete flag */
}
```

AD_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 AD_ComparatorOff after the process of this API function completes.

[Classification]

CG_ad.c

[Syntax]

```
void    AD_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

AD_SelectADChannel

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]

CG_ad.c

[Syntax]

- [Fx3] [Ix3] [Kx3-A] [Kx3-L] [Lx3]

```
#include    "CG_ad.h"
MD_STATUS   AD_SelectADChannel ( enum ADChannel channel );
```

- [Kx3]

```
#include    "CG_ad.h"
void    AD_SelectADChannel ( 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 CG_ad.h for details about the analog voltage input pin ADCHANNEL*n*.

[Return value]

- [Fx3] [Ix3] [Kx3-A] [Kx3-L] [Lx3]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

- [Kx3]

None.

AD_Read

Reads the results of A/D conversion.

Remark If the target device is 78K0R/Fx3, 78K0R/Kx3, 78K0R/Kx3-L, then the results of A/D conversion will be 10 bits. If the target device is 78K0R/Kx3-A, 78K0R/Lx3, then the results of A/D conversion will be 12 bits.

[Classification]

CG_ad.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    AD_Read ( USHORT *buffer );
```

[Argument(s)]

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

[Return value]

None.

AD_ReadByte

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

[Classification]

CG_ad.c

[Syntax]

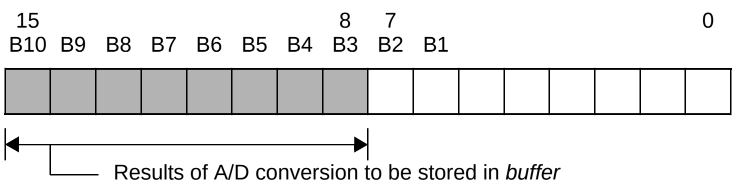
```
#include    "CG_macrodriver.h"
void    AD_ReadByte ( UCHAR *buffer );
```

[Argument(s)]

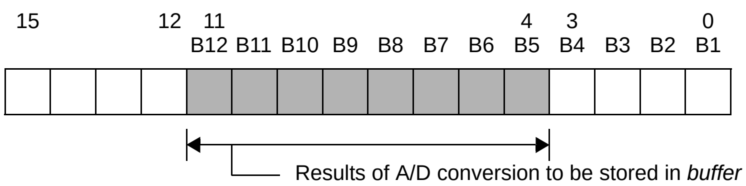
I/O	Argument	Description
O	`UCHAR *buffer;`	Pointer to area in which to store the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution)

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

- [Fx3] [Ix3] [Kx3] [Kx3-L]



- [Kx3-A] [Lx3]



[Return value]

None.

C.3.9 D/A Converter

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

Table C-10. API Functions: [D/A Converter]

API Function Name	Function
DA_Init	Performs initialization necessary to control D/A converter functions.
DA_UserInit	Performs user-defined initialization relating to the D/A converter.
DA_PowerOff	Halts the clock supplied to the D/A converter.
DAn_Start	Starts D/A conversion.
DAn_Stop	Ends D/A conversion.
DAn_SetValue	Sets the initial analog voltage output to the ANO*n* pin.
DAn_Set8BitsValue	Sets the initial analog voltage (8 bits) output to the ANO*n* pin.
DAn_Set12BitsValue	Sets the initial analog voltage (12 bits) output to the ANO*n* pin.

DA_Init

Performs initialization necessary to control D/A converter functions.

[Classification]

CG_da.c

[Syntax]

```
void    DA_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

DA_UserInit

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

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

[Classification]

CG_da_user.c

[Syntax]

```
void    DA_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

DA_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 (e.g. D/A converter mode register: DAM) after this API function is called are ignored.

[Classification]

CG_da.c

[Syntax]

```
void    DA_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

DA*n*_Start

Starts D/A conversion.

[Classification]

CG_da.c

[Syntax]

```
void    DAn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DA*n*_Stop

Ends D/A conversion.

[Classification]

CG_da.c

[Syntax]

```
void    DAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DA*n*_SetValue

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

[Classification]

CG_da.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    DAn_SetValue ( UCHAR value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`UCHAR value;`	Analog voltage (0x0 to 0xff)

[Return value]

None.

[Example]

Below is an example of setting "analog voltage" to channels 0 and 1.

[CG_main.c]

```
void main ( void ) {
    ......
    DA0_Start ();                       /* Start D/A conversion */
    DA1_Start ();                       /* Start D/A conversion */
    ......
    DA0_SetValue ( 0x7f );              /* Set analog voltage */
    ......
}
```

[CG_timer_user.c]

```
#include    "CG_macrodriver.h"
UCHAR   gValue = 0;
__interrupt void MD_INTTM05 ( void ) {  /* Interrupt processing for INTTM05 */
    DA1_SetValue ( gValue++ );          /* Set analog voltage */
}
```

DA*n*_Set8BitsValue

Sets the analog voltage (8 bits) output to the ANO*n* pin.

[Classification]

CG_da.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    DAn_Set8BitsValue ( UCHAR value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *value*;	Analog voltage (0x0 to 0x1f)

[Return value]

None.

[Example]

Below is an example of setting "analog voltage" to channels 0 and 1.

[CG_main.c]

```
void main ( void ) {
    ......
    DA0_Start ();                       /* Start D/A conversion */
    DA1_Start ();                       /* Start D/A conversion */
    ......
    DA0_Set8BitsValue ( 0x7f );         /* Set analog voltage */
    ......
}
```

[CG_timer_user.c]

```
#include    "CG_macrodriver.h"
UCHAR   gValue = 0;
__interrupt void MD_INTTM05 ( void ) {  /* Interrupt processing for INTTM05 */
    DA1_Set8BitsValue ( gValue++ );     /* Set analog voltage */
}
```

DA*n*_Set12BitsValue

Sets the analog voltage (12 bits) output to the ANO*n* pin.

[Classification]

CG_da.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    DAn_Set12BitsValue ( UCHAR value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *value*;	Analog voltage (0x0 to 0xfff)

[Return value]

None.

[Example]

Below is an example of setting "analog voltage" to channels 0 and 1.

[CG_main.c]

```
void main ( void ) {
    ......
    DA0_Start ();                       /* Start D/A conversion */
    DA1_Start ();                       /* Start D/A conversion */
    ......
    DA0_Set12BitsValue ( 0x1ff );       /* Set analog voltage */
    ......
}
```

[CG_timer_user.c]

```
#include    "CG_macrodriver.h"
UCHAR   gValue = 0;
__interrupt void MD_INTTM05 ( void ) {  /* Interrupt processing for INTTM05 */
    DA1_Set12BitsValue ( gValue++ );    /* Set analog voltage */
}
```

C.3.10 Timer

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

Table C-11. API Functions: [Timer]

API Function Name	Function
TAUm_Init	Performs initialization necessary to control timer array unit functions.
TAUm_UserInit	Performs user-defined initialization relating to the timer array unit.
TAUm_PowerOff	Halts the clock supplied to the timer array unit.
TAUm_Channeln_Start	Starts the count for channel *n*.
TAUm_Channeln_Stop	Ends the count for channel *n*.
TAUm_Channeln_ChangeCondition	Changes the counter value.
TAUm_Channeln_ChangeTimerCondition	Changes the counter value.
TAUm_Channeln_GetPulseWidth	Captures the high/low-level width measured between pulses of the signal (pulses) input to the TI*mn* pin.
TAUm_Channeln_ChangeDuty	Changes the duty ratio of the PWM signal output to the TO*mn* pin.
TAUm_Channeln_SoftWareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.

TAU*m*_Init

Performs initialization necessary to control timer array unit functions.

[Classification]

CG_timer.c

[Syntax]

```
void    TAUm_Init ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

TAU*m*_UserInit

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

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

[Classification]

CG_timer_user.c

[Syntax]

```
void    TAUm_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

TAU*m*_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 (e.g. timer clock select register 0: TPS0) after this API function is called are ignored.

[Classification]

CG_timer.c

[Syntax]

```
void    TAUm_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

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]

CG_timer.c

[Syntax]

```
void    TAUm_Channeln_Start ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

TAU*m*_Channel*n*_Stop

Ends the count for channel *n*.

[Classification]

CG_timer.c

[Syntax]

```
void    TAUm_Channeln_Stop ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

TAU*m*_Channel*n*_ChangeCondition

Changes the counter value.

Remarks 1. The value specified in parameter *regvalue* is set to timer data register *mn* (TDR*mn*).

2. The timing for calling these API functions differs as follows, depending on the type of function (e.g. interval timer, square wave output, or external event counter).

Function Type	Timing for Calling
Interval timer	Can call at user discretion.
Square wave output	Can call at user discretion.
Divider function	Can call at user discretion.
External event counter	Can call at user discretion.
Input pulse interval measurement	Cannot be called.
Input pulse high-/low-level width measurement	Cannot be called.
PWM output	Cannot be called.
One-shot pulse output	Cannot be called during operation.
Multiple PWM output	Cannot be called.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TAUm_Channeln_ChangeCondition ( USHORT regvalue );
```

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

[Argument(s)]

I/O	Argument	Description
I	`USHORT regvalue;`	Counter value (0x0 to 0xffff)

[Return value]

None.

[Example]

The example below shows changing the interval time to one half.
In this example, channel 0 has been selected for the interval timer.

[CG_main.c]

```
#include    "CG_macrodriver.h"
```

```
void main ( void ) {
    USHORT  value = TAU_TDR00_VALUE >> 1;      /* TAU_TDR00_VALUE: Current interval time */
    ......
    TAU0_Channel0_Start ();                      /* Start count */
    ......
    TAU0_Channel0_ChangeCondition ( value );     /* Change counter value */
   ......
}
```

TAU*m*_Channel*n*_ChangeTimerCondition

Changes the counter value.

Remarks 1. The value specified in parameter *regvalue* is set to timer data register *mn* (TDR*mn*).

2. The timing for calling these API functions differs as follows, depending on the type of function (e.g. interval timer, square wave output, or external event counter).

Function Type	Timing for Calling
Interval timer	Can call at user discretion.
Square wave output	Can call at user discretion.
Divider function	Can call at user discretion.
External event counter	Can call at user discretion.
Input pulse interval measurement	Cannot be called.
Input pulse high-/low-level width measurement	Cannot be called.
PWM output	Cannot be called.
One-shot pulse output	Cannot be called during operation.
Multiple PWM output	Cannot be called.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TAUm_Channeln_ChangeTimerCondition ( USHORT regvalue );
```

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

[Argument(s)]

I/O	Argument	Description
I	USHORT *regvalue*;	Counter value (0x0 to 0xffff)

[Return value]

None.

[Example]

The example below shows changing the interval time to one half.
In this example, channel 0 has been selected for the interval timer.

[CG_main.c]

```
#include    "CG_macrodriver.h"
```

```
void main ( void ) {
    USHORT  value = TAU_TDR00_VALUE >> 1;      /* TAU_TDR00_VALUE: Current interval time */
    ......
    TAU0_Channel0_Start ();                        /* Start count */
    ......
    TAU0_Channel0_ChangeTimerCondition ( value );  /* Change counter value */
   ......
}
```

TAU*m*_Channel*n*_GetPulseWidth

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

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TAUm_Channeln_GetPulseWidth ( ULONG *width );
```

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

[Argument(s)]

I/O	Argument	Description
O	`ULONG *width;`	Pointer to an area to store the measurement width (0x0 to 0x1ffff)

[Return value]

None.

TAU*m*_Channel*n*_ChangeDuty

Changes the duty ratio of the PWM signal output to the TO*mn* pin.

Remark The timing for calling these API functions differs as follows, depending on the type of function (e.g. interval timer, square wave output, or external event counter).

Function Type	Timing for Calling
Interval timer	Cannot be called.
Square wave output	Cannot be called.
Divider function	Cannot be called.
External event counter	Cannot be called.
Input pulse interval measurement	Cannot be called.
Input pulse high-/low-level width measurement	Cannot be called.
PWM output	After INTTM*mn* master channel interrupt.
One-shot pulse output	Cannot be called.
Multiple PWM output	After INTTM*mn* master channel interrupt.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TAUm_Channeln_ChangeDuty ( UCHAR ratio );
```

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

[Argument(s)]

I/O	Argument	Description
I	UCHAR *ratio*;	Duty ratio (0 to 100, unit: %)

Remark The value set to duty ratio *ratio* must be in base 10 notation.

[Return value]

None.

[Example]

The example below shows changing the duty ratio to 25%.
In this example, channels 0 and 1 have been selected for PWM output or multiplex PWM output.

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    UCHAR   ratio = 25;
    ......
    TAU0_Channel0_Start ();             /* Start count */
    ......
    TAU0_Channel1_ChangeDuty ( ratio ); /* Change duty ratio */
    ......
}
```

TAU*m*_Channel*n*_SoftWareTriggerOn

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

[Classification]

CG_timer.c

[Syntax]

```
void    TAUm_Channeln_SoftWareTriggerOn ( void );
```

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

[Argument(s)]

None.

[Return value]

None.

C.3.11 Watchdog Timer

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

Table C-12. API Functions: [Watchdog Timer]

API Function Name	Function
WDT_Init	Performs initialization necessary to control watchdog timer functions.
WDT_UserInit	Performs user-defined initialization relating to the watchdog timer.
WDT_Restart	Clears the watchdog timer counter and resumes counting.

WDT_Init

Performs initialization necessary to control watchdog timer functions.

[Classification]

CG_wdt.c

[Syntax]

```
void    WDT_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

WDT_UserInit

Performs user-defined initialization relating to the watchdog timer.

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

[Classification]

CG_wdt_user.c

[Syntax]

```
void    WDT_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

WDT_Restart

Clears the watchdog timer counter and resumes counting.

[Classification]

CG_wdt.c

[Syntax]

```
void    WDT_Restart ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.12 Real-time Clock

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

Table C-13. API Functions: [Real-time Clock]

API Function Name	Function
RTC_Init	Performs initialization necessary to control real-time counter functions.
RTC_UserInit	Performs user-defined initialization relating to the real-time counter.
RTC_PowerOff	Halts the clock supplied to the real-time counter.
RTC_CounterEnable	Starts the count of the real-time counter (year, month, weekday, day, hour, minute, second).
RTC_CounterDisable	Ends the count of the real-time counter (year, month, weekday, day, hour, minute, second).
RTC_SetHourSystem	Sets the clock type (12-hour or 24-hour clock) of the real-time counter.
RTC_CounterSet	Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
RTC_CounterGet	Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
RTC_ConstPeriodInterruptEnable	Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.
RTC_ConstPeriodInterruptDisable	Ends the cyclic interrupt function.
RTC_ConstPeriodInterruptCallback	Performs processing in response to the cyclic interrupt INTRTC.
RTC_AlarmEnable	Starts the alarm interrupt function.
RTC_AlarmDisable	Ends the alarm interrupt function.
RTC_AlarmSet	Sets the alarm conditions (weekday, hour, minute).
RTC_AlarmGet	Reads the alarm conditions (weekday, hour, minute).
RTC_AlarmInterruptCallback	Performs processing in response to the alarm interrupt INTRTC.
RTC_IntervalStart	Starts the interval interrupt function.
RTC_IntervalStop	Ends the interval interrupt function.
RTC_IntervalInterruptEnable	Sets the cycle of the interrupts INTRTCI, then starts the interval interrupt function.
RTC_IntervalInterruptDisable	Ends the interval interrupt function.
RTC_RTC1HZ_OutputEnable	Enables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
RTC_RTC1HZ_OutputDisable	Disables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
RTC_RTCCL_OutputEnable	Enables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
RTC_RTCCL_OutputDisable	Disables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
RTC_RTCDIV_OutputEnable	Enables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
RTC_RTCDIV_OutputDisable	Disables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
RTC_ChangeCorrectionValue	Changes the timing and correction value for correcting clock errors.

RTC_Init

Performs initialization necessary to control real-time counter functions.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_UserInit

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

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

[Classification]

CG_rtc_user.c

[Syntax]

```
void    RTC_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_PowerOff

Halts the clock supplied to the real-time counter.

Remark Calling this API function changes the real-time counter to reset status. For this reason, writes to the control registers (e.g. real-time counter control register 0: RTCC0) after this API function is called are ignored.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_CounterEnable

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

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_CounterEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_CounterDisable

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

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_CounterDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_SetHourSystem

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

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_SetHourSystem ( 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 "CG_rtc.h" larger.

[Example]

Below is an example of setting the clock type to the 24-hour clock.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    ......
    RTC_CounterEnable ();           /* Start count */
    ......
    RTC_SetHourSystem ( HOUR24 );   /* Set clock type */
    ......
```

```
}
```

RTC_CounterSet

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

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_CounterSet ( 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 counter.

```
struct  RTCCounterValue {
    UCHAR   Sec;    /* second */
    UCHAR   Min;    /* Minute */
    UCHAR   Hour;   /* Hour */
    UCHAR   Day;    /* Day */
    UCHAR   Week;   /* Weekday (0: Sunday, 6: Saturday) */
    UCHAR   Month;  /* Month */
    UCHAR   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 "CG_rtc.h" larger.

[Example]

The example below shows the counter value of the real-time counter being set to "2008/12/25 (Thu.) 17:30:00".

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( main ) {
    struct  RTCCounterValue counterwriteval;
    ......
    RTC_CounterEnable ();               /* Start count */
    ......
    counterwriteval.Year = 0x08;
    counterwriteval.Month = 0x12;
    counterwriteval.Day = 0x25;
    counterwriteval.Week = 0x05;
    counterwriteval.Hour = 0x17;
    counterwriteval.Min = 0x30;
    counterwriteval.Sec = 0;
    RTC_SetHourSystem ( HOUR24 );       /* Set clock type */
    RTC_CounterSet ( counterwriteval ); /* Set counter value */
    ......
}
```

RTC_CounterGet

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

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_CounterGet ( 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 RTC_CounterSet 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 "CG_rtc.h" larger.

[Example]

Below is an example of reading the counter value of the real-time counter.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCCounterValue counterreadval;
    ......
    RTC_CounterEnable ();              /* Start count */
    ......
    RTC_CounterGet ( &counterreadval ); /* Read count value */
    ......
```

```
}
```

RTC_ConstPeriodInterruptEnable

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

[Classification]

CG_rtc.c

[Syntax]

```
#include     "CG_rtc.h"
MD_STATUS    RTC_ConstPeriodInterruptEnable ( 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

[Example]

Below is an example of setting the cycle of the interrupts INTRTC, then starting the cyclic interrupt function.

[CG_main.c]

```
#include     "CG_rtc.h"
void main ( void ) {
    ......
    RTC_ConstPeriodInterruptDisable ();         /* End of cyclic interrupt function */
    ......
    RTC_ConstPeriodInterruptEnable ( HALFSEC ); /* Start of cyclic interrupt function */
    ......
}
```

RTC_ConstPeriodInterruptDisable

Ends the cyclic interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_ConstPeriodInterruptDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_ConstPeriodInterruptCallback

Performs processing in response to the cyclic interrupt INTRTC.

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

[Classification]

CG_rtc_user.c

[Syntax]

```
void    RTC_ConstPeriodInterruptCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmEnable

Starts the alarm interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_AlarmEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmDisable

Ends the alarm interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_AlarmDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmSet

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

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_rtc.h"
void    RTC_AlarmSet ( struct RTCAlarmValue alarmval );
```

[Argument(s)]

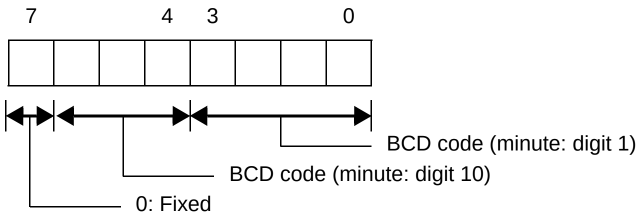
I/O	Argument	Description
I	`struct RTCAlarmValue alarmval;`	Alarm conditions (weekday, hour, minute)

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

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

- Alarmwm (Minute)

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

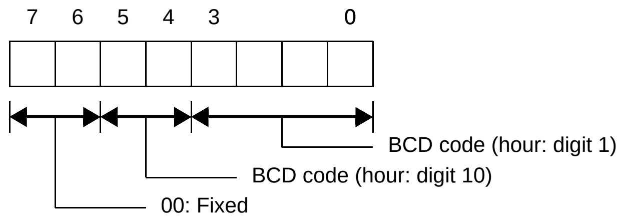


- Alarmwh (Hour)

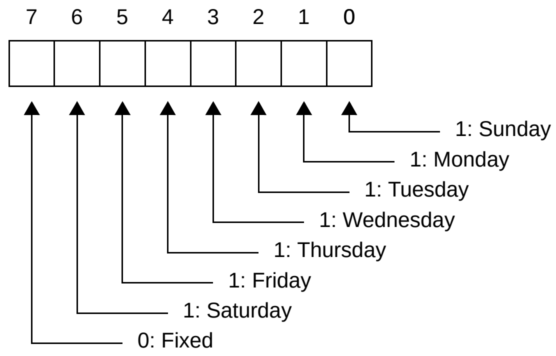
  Below are shown the meanings of each bit of the structure member Alarmwh. If the real-time counter is set to the 12-hour clock, then bit 5 has the following meaning.

  0: AM

  1: PM



- Alarmww (Weekday)
  Below are shown the meanings of each bit of the structure member Alarmww.



[Return value]

None.

[Example 1]

The example below shows the alarm conditions being set to "Monday/Tuesday/Wednesday at 17:30".

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ......
    RTC_AlarmEnable ();             /* Start alarm interrupt function */
    RTC_CounterEnable ();           /* Start count */
    ......
    RTC_SetHourSystem ( HOUR24 );   /* Set clock type */
    alarmval.Alarmww = 0xe;
    alarmval.Alarmwh = 0x17;
    alarmval.Alarmwm = 0x30;
    RTC_AlarmSet ( alarmval );      /* Set conditions */
    ......
}
```

[Example 2]

The example below shows the alarm conditions being set to "Saturday/Sunday (time left unchanged)".

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ......
    RTC_AlarmEnable ();             /* Start alarm interrupt function */
    ......
    alarmval.Alarmww = 0x41;
    RTC_AlarmSet ( alarmval );      /* Change conditions */
    ......
}
```

RTC_AlarmGet

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

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_rtc.h"
void    RTC_AlarmGet ( struct RTCAlarmValue *alarmval );
```

Remark See RTC_AlarmSet 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.

[Example]

The example below shows the alarm conditions being read.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ......
    RTC_AlarmEnable ();         /* Start alarm interrupt function */
    ......
    RTC_AlarmGet ( &alarmval ); /* Read conditions */
    ......
}
```

RTC_AlarmInterruptCallback

Performs processing in response to the alarm interrupt INTRTC.

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

[Classification]

CG_rtc_user.c

[Syntax]

```
void    RTC_AlarmInterruptCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_IntervalStart

Starts the interval interrupt function.

Remark After setting the cycle of the interrupts INTRTCI, call RTC_IntervalInterruptEnable to start the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_IntervalStart ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_IntervalStop

Ends the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_IntervalStop ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_IntervalInterruptEnable

Sets the cycle of the interrupts INTRTCI, then starts the interval interrupt function.

Remark Call RTC_IntervalStart to start the interval interrupt function without setting the cycle of the interrupts INTRTCI.

[Classification]

CG_rtc.c

[Syntax]

- [Ix3 (excluding IB3)] [Kx3-A] [Kx3-L] [Lx3]

```
#include    "CG_rtc.h"
MD_STATUS   RTC_IntervalInterruptEnable ( enum RTCINTInterval interval );
```

- [Kx3]

```
#include    "CG_rtc.h"
void        RTC_IntervalInterruptEnable ( enum RTCINTInterval interval );
```

[Argument(s)]

I/O	Argument	Description
I	`enum RTCINTInterval interval;`	Interrupt INTRTCI cycle INTERVAL0: 2^6/fXT INTERVAL1: 2^7/fXT INTERVAL2: 2^8/fXT INTERVAL3: 2^9/fXT INTERVAL4: 2^10/fXT INTERVAL5: 2^11/fXT INTERVAL6: 2^12/fXT

Remark fXT is the frequency of the subsystem clock.

[Return value]

- [Ix3 (excluding IB3)] [Kx3-A] [Kx3-L] [Lx3]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

- [Kx3]

None.

[Example]

Below is an example of changing the interval, the restarting the interval interrupt function.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    ......
    RTC_IntervalStart ();                      /* Start interval interrupt function */
    ......
    RTC_IntervalStop ();                       /* End interval interrupt function */
    ......
    RTC_IntervalInterruptEnable ( INTERVAL6 );  /* Start interval interrupt function */
    ......
}
```

RTC_IntervalInterruptDisable

Ends the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_IntervalInterruptDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RTC1HZ_OutputEnable

Enables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTC1HZ_OutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RTC1HZ_OutputDisable

Disables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTC1HZ_OutputDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RTCCL_OutputEnable

Enables output of the real-time counter clock (32 kHz source) to the RTCCL pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTCCL_OutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RTCCL_OutputDisable

Disables output of the real-time counter clock (32 kHz source) to the RTCCL pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTCCL_OutputDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RTCDIV_OutputEnable

Enables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTCDIV_OutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RTCDIV_OutputDisable

Disables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTCDIV_OutputDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_ChangeCorrectionValue

Changes the timing and correction value for correcting clock errors.

[Classification]

CG_rtc.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_ChangeCorrectionValue ( enum RTCCorectionTiming timing, UCHAR corectVal );
```

[Argument(s)]

I/O	Argument	Description
I	`enum RTCCorectionTiming timing;`	When clock errors are corrected EVERY20S: When the seconds digits are 00, 20 or 40 EVERY60S: When the seconds digits are 00
I	`UCHAR corectVal;`	Clock error correction value

Remark This API function does not correct clock errors if correction value *corectVal* is set to 0x0, 0x1, 0x40 or 0x41.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

C.3.13 Clock Output

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

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

API Function Name	Function
PCL_Init	Performs initialization necessary to control clock output control circuit functions.
PCL_UserInit	Performs user-defined initialization relating to the clock output control circuits.
PCL_Start	Starts clock output.
PCL_Stop	Ends clock output.
PCL_ChangeFreq	Changes the output clock to the PCL pin.

PCL_Init

Performs initialization necessary to control clock output control circuit functions.

[Classification]

CG_pcl.c

[Syntax]

```
void    PCL_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

PCL_UserInit

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

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

[Classification]

CG_pcl_user.c

[Syntax]

```
void    PCL_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

PCL_Start

Starts clock output.

[Classification]

CG_pcl.c

[Syntax]

```
void    PCL_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

PCL_Stop

Ends clock output.

[Classification]

CG_pcl.c

[Syntax]

```
void    PCL_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

PCL_ChangeFreq

Changes the output clock to the PCL pin.

Remark The value specified in parameter *clock* is set to clock output select register (CKS).

[Classification]

CG_pcl.c

[Syntax]

```
#include    "CG_pclbuz.h"
void    PCL_ChangeFreq ( enum PCLclock clock );
```

[Argument(s)]

I/O	Argument	Description
I	`enum PCLclock clock;`	Output clock type MAINCLOCK: fMAIN MAIN2: fMAIN/2 MAIN4: fMAIN/4 MAIN8: fMAIN/8 MAIN16: fMAIN/16 MAIN2048: fMAIN/2048 MAIN4096: fMAIN/4096 MAIN8192: fMAIN/8192 PLLCLOCK: fPLL PLL2: fPLL/2 PLL4: fPLL/4 PLL8: fPLL/8 PLL16: fPLL/16 PLL2048: fPLL/2048 PLL4096: fPLL/4096 PLL8192: fPLL/8192 ILCLOCK: fIL SUBCLOCK: fSUB

Remark fMAIN is the main system clock frequency; fPLL is the PLL clock frequency; fIL is the internal low-speed oscillation clock frequency; fSUB is the subsystem clock frequency.

[Return value]

None.

C.3.14 Clock Output/Buzzer Output

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

Table C-15. API Functions: [Clock Output/Buzzer Output]

API Function Name	Function
PCLBUZn_Init	Performs initialization necessary to control clock/buzzer output control circuit functions.
PCLBUZn_UserInit	Performs user-defined initialization relating to the clock/buzzer output control circuits.
PCLBUZn_Start	Starts clock/buzzer output.
PCLBUZn_Stop	Ends clock/buzzer output.
PCLBUZn_ChangeFreq	Changes the output clock to the PCLBUZ*n* pin.

PCLBUZ*n*_Init

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

[Classification]

CG_pclbuz.c

[Syntax]

```
void    PCLBUZn_Init ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

PCLBUZ*n*_UserInit

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

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

[Classification]

CG_pclbuz_user.c

[Syntax]

```
void    PCLBUZn_UserInit ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

PCLBUZ*n*_Start

Starts clock/buzzer output.

[Classification]

CG_pclbuz.c

[Syntax]

```
void    PCLBUZn_Start ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

PCLBUZ*n*_Stop

Ends clock/buzzer output.

[Classification]

CG_pclbuz.c

[Syntax]

```
void    PCLBUZn_Stop ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

PCLBUZ*n*_ChangeFreq

Changes the output clock to the PCLBUZ*n* pin.

Remark The value specified in parameter *clock* is set to clock output select register *n* (CKS*n*).

[Classification]

CG_pclbuz.c

[Syntax]

```
#include     "CG_pclbuz.h"
MD_STATUS    PCLBUZn_ChangeFreq ( enum PCLBUZclock clock );
```

Remark *n* is the output pin.

[Argument(s)]

I/O	Argument	Description
I	`enum PCLBUZclock clock;`	Output clock type MAINCLOCK: fMAIN MAIN2: fMAIN/2 MAIN4: fMAIN/4 MAIN8: fMAIN/8 MAIN16: fMAIN/16 MAIN2048: fMAIN/2048 MAIN4096: fMAIN/4096 MAIN8192: fMAIN/8192 SUBCLOCK: fSUB SUB2: fSUB/2 SUB4: fSUB/4 SUB8: fSUB/8 SUB16: fSUB/16 SUB32: fSUB/32 SUB64: fSUB/64 SUB128: fSUB/128

Remark fMAIN is the main system clock frequency; fSUB is the subsystem clock frequency.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of changing the output clock of pin PCLBUZ0 each time an externally connected button is pressed, in a system that generates the interrupt INTP0 each time the button is pressed.

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_pclbuz.h"
BOOL    gFlag;                                      /* Button pressed flag */
#define PCLBUZ_FREQUENCY    8                       /* Total number of output clock types */
const   enum    PCLBUZclock gClock[PCLBUZ_FREQUENCY] = {    /* Output clock type */
    PCLBUZ_OUTCLK_fSUB0, PCLBUZ_OUTCLK_fSUB1, PCLBUZ_OUTCLK_fSUB2,
    PCLBUZ_OUTCLK_fSUB3, PCLBUZ_OUTCLK_fSUB4, PCLBUZ_OUTCLK_fSUB5,
    PCLBUZ_OUTCLK_fSUB6, PCLBUZ_OUTCLK_fSUB7
};
void main ( void ) {
    int     index = 0;
    gFlag = 0;                                      /* Initialize button pressed flag */
    ......
    PCLBUZ0_Start ();                               /* Start clock/buzzer output */
    while ( 1 ) {
        if ( gFlag ) {                              /* Wait for INTP0 */
            PCLBUZ_ChangeFreq ( gClock[index++] );  /* Change output clock */
            if ( index >= PCLBUZ_FREQUENCY ) {
                index = 0;
            }
            gFlag = 0;                              /* Clear button pressed flag */
        }
    }
}
```

[CG_int_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* Button pressed flag */
__interrupt void MD_INTP0 ( void ) {        /* Interrupt processing for INTP0 */
    gFlag = 1;                              /* Set button pressed flag */
}
```

C.3.15 LCD Controller/Driver

Below is a list of API functions output by Code Generator for LCD controller/driver use.

Table C-16. API Functions: [LCD Controller/Driver]

API Function Name	Function
LCD_Init	Performs initialization necessary to control LCD controller/driver functions.
LCD_UserInit	Performs user-defined initialization relating to the LCD controller/driver.
LCD_DisplayOn	Sets the LCD controller/driver to "display on" status.
LCD_DisplayOff	Sets the LCD controller/driver to "display off" status.
LCD_VoltageOn	Enables operation of the LCD controller/driver's voltage boost circuit and capacitor split circuit, then outputs the deselect signal from the segment pin.
LCD_VoltageOff	Halts operation of the LCD controller/driver's voltage boost circuit and capacitor split circuit, then outputs the groundlevel signal from the segment/common pin.

LCD_Init

Performs initialization necessary to control LCD controller/driver functions.

[Classification]

CG_lcd.c

[Syntax]

```
void    LCD_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

LCD_UserInit

Performs user-defined initialization relating to the LCD controller/driver.

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

[Classification]

CG_lcd_user.c

[Syntax]

```
void    LCD_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

LCD_DisplayOn

Sets the LCD controller/driver to "display on" status.

[Classification]

CG_lcd.c

[Syntax]

```
void    LCD_DisplayOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

LCD_DisplayOff

Sets the LCD controller/driver to "display off" status.

[Classification]

CG_lcd.c

[Syntax]

```
void    LCD_DisplayOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

LCD_VoltageOn

Enables operation of the LCD controller/driver's voltage boost circuit and capacitor split circuit, then outputs the deselect signal from the segment pin.

[Classification]

CG_lcd.c

[Syntax]

```
void    LCD_VoltageOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

LCD_VoltageOff

Halts operation of the LCD controller/driver's voltage boost circuit and capacitor split circuit, then outputs the ground-level signal from the segment/common pin.

[Classification]

CG_lcd.c

[Syntax]

```
void    LCD_VoltageOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.16 DMA

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

Table C-17. API Functions: [DMA]

API Function Name	Function
DMAn_Init	Performs initialization necessary to control DMA controller functions.
DMAn_UserInit	Performs user-defined initialization relating to the DMA controller.
DMAn_Enable	Enables operation of channel *n*.
DMAn_Disable	Disables operation of channel *n*.
DMAn_Hold	Holds a DMA start request.
DMAn_Restart	Releases hold on a DMA start request.
DMAn_CheckStatus	Reads the transfer status (transfer complete/transfer ongoing).
DMAn_SetData	Sets the RAM address of the transfer source/destination, and the number of times the data has been transferred.
DMAn_SoftwareTriggerOn	Starts DMA transfer when DMA operation is enabled.

DMA*n*_Init

Performs initialization necessary to control DMA controller functions.

[Classification]

CG_dma.c

[Syntax]

```
void    DMAn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA*n*_UserInit

Performs user-defined initialization relating to the DMA controller.

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

[Classification]

CG_dma_user.c

[Syntax]

```
void    DMAn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA*n*_Enable

Enables operation of channel *n*.

[Classification]

CG_dma.c

[Syntax]

```
void    DMAn_Enable ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA*n*_Disable

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 via DMAn_CheckStatus.

[Classification]

CG_dma.c

[Syntax]

```
void    DMAn_Disable ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

[Example]

The example below shows setting the operation mode of channel 0 to "disabled".

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    while ( MD_COMPLETED == DMA0_CheckStatus () ); /* Check transfer status */
    DMA0_Disable ();                               /* Change to operation disabled status */
    ......
}
```

DMA*n*_Hold

Holds a DMA start request.

Remark Call DMAn_Restart to release the hold on DMA start requests.

[Classification]

CG_dma.c

[Syntax]

```
void    DMAn_Hold ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA*n*_Restart

Releases hold on a DMA start request.

[Classification]

CG_dma.c

[Syntax]

```
void    DMAn_Restart ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA*n*_CheckStatus

Reads the transfer status (transfer complete/transfer ongoing).

[Classification]

CG_dma.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   DMAn_CheckStatus ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

Macro	Description
MD_UNDEREXEC	Transfer ongoing
MD_COMPLETED	Transfer complete

DMA*n*_SetData

Sets the RAM address of the transfer source/destination, and the number of times the data has been transferred.

Remark Calling this API function while a transfer is ongoing will end the transfer.

[Classification]

CG_dma.c

[Syntax]

- [Fx3] [Ix3] [Kx3-A] [Kx3-L] [Lx3]

```
#include     "CG_macrodriver.h"
MD_STATUS    DMAn_SetData ( USHORT sfraddr, USHORT ramaddr, USHORT count );
```

- [Kx3]

```
#include     "CG_macrodriver.h"
MD_STATUS    DMAn_SetData ( USHORT ramaddr, USHORT count );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`USHORT` *`sfrcaddr`*`;`	SFR address of source/destination
I	`USHORT` *`ramcaddr`*`;`	RAM address of source/destination
I	`USHORT` *`count`*`;`	Number of data transmissions (1 to 1024)

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

DMA*n*_SoftwareTriggerOn

Starts DMA transfer when DMA operation is enabled.

[Classification]

CG_dma.c

[Syntax]

```
void    DMAn_SoftwareTriggerOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

[Example]

Below is an example of software trigger as a DMA transfer start trigger.

[CG_main.c]

```
void main ( void ) {
    ......
    DMA0_Enable ();             /* Change to operation enabled status */
    DMA0_SoftwareTriggerOn ();  /* Start DMA transfer */
    ......
}
```

C.3.17 LVI

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

Table C-18. API Functions: [LVI]

API Function Name	Function
LVI_Init	Performs initialization necessary to control low-voltage detector functions.
LVI_UserInit	Performs user-defined initialization relating to the low-voltage detector.
LVI_InterruptModeStart	Starts low-voltage detection (when in interrupt generation mode).
LVI_ResetModeStart	Starts low-voltage detection (when in internal reset mode).
LVI_Stop	Stops low-voltage detection.
LVI_SetLVILevel	Sets the low-voltage detection level.

LVI_Init

Performs initialization necessary to control low-voltage detector functions.

[Classification]

CG_lvi.c

[Syntax]

```
void    LVI_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

LVI_UserInit

Performs user-defined initialization relating to the low-voltage detector.

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

[Classification]

CG_lvi_user.c

[Syntax]

```
void    LVI_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

LVI_InterruptModeStart

Starts low-voltage detection (when in interrupt generation mode).

[Classification]

CG_lvi.c

[Syntax]

```
void    LVI_InterruptModeStart ( void );
```

[Argument(s)]

None.

[Return value]

None.

[Example]

The example below shows the detection of low voltage when the operation mode is interrupt generation mode (generate the interrupt INTLVI).

[CG_main.c]

```
void main ( void ) {
    ......
    LVI_InterruptModeStart ( );         /* Start low-voltage detection */
    ......
}
```

[CG_lvi_user.c]

```
__interrupt void MD_INTLVI ( void ) {   /* Interrupt processing for INTLVI */
    if ( LVIF == 1 ) {                  /* Trigger identification: Check LVIF flag */
        ...... /* Handle case when "power voltage (VDD) < detected voltage (VLVI)" detected */
    } else {
        ...... /* Handle case when "power voltage (VDD) >= detected voltage (VLVI)" detected */
    }
}
```

LVI_ResetModeStart

Starts low-voltage detection (when in internal reset mode).

[Classification]

CG_lvi.c

[Syntax]

```
MD_STATUS   LVI_ResetModeStart ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend) - The program is configured to not use the low-voltage detector function. - The object of low voltage detection is external voltage (V_{DD}), and power voltage (V_{DD}) <= detected voltage (V_{LVI}). - The object of low voltage detection is external input voltage (EXLVI), and external input voltage (EXLVI) <= detected voltage (V_{EXLVI}).

LVI_Stop

Stops low-voltage detection.

[Classification]

CG_lvi.c

[Syntax]

```
void    LVI_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

LVI_SetLVILevel

Sets the low-voltage detection level.

Remarks 1. To change the low-voltage detection level, you must call LVI_Stop before calling this API function.

2. The value specified in parameter *level* is set to low-voltage detection level select register (LVIS).

[Classification]

CG_lvi.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_lvi.h"
MD_STATUS   LVI_SetLVILevel ( enum LVILevel level );
```

[Argument(s)]

I/O	Argument	Description
I	`enum LVILevel level;`	Voltage level to detect as low voltage LVILEVEL0: 4.22 V ± 0.1 V LVILEVEL1: 4.07 V ± 0.1 V LVILEVEL2: 3.92 V ± 0.1 V LVILEVEL3: 3.76 V ± 0.1 V LVILEVEL4: 3.61 V ± 0.1 V LVILEVEL5: 3.45 V ± 0.1 V LVILEVEL6: 3.30 V ± 0.1 V LVILEVEL7: 3.15 V ± 0.1 V LVILEVEL8: 2.99 V ± 0.1 V LVILEVEL9: 2.84 V ± 0.1 V LVILEVEL10: 2.68 V ± 0.1 V LVILEVEL11: 2.53 V ± 0.1 V LVILEVEL12: 2.38 V ± 0.1 V LVILEVEL13: 2.22 V ± 0.1 V LVILEVEL14: 2.07 V ± 0.1 V LVILEVEL15: 1.91 V ± 0.1 V

Remark If the target device is 78K0R/Fx3 or 78K0R/Ix3, then keyword that can be specified for *level* is limited to "LVILEVEL0" to "LVILEVEL9".

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend)
MD_ARGERROR	Invalid argument specification

APPENDIX D INDEX

T

U

Revision Record

Rev.	Date	Description	
		Page	**Summary**
1.00	Sep 01, 2012	-	First Edition issued

CubeSuite+ V1.03.00 User's Manual:
78K0R Design

Publication Date: Rev.1.00 Sep 01, 2012

Published by: Renesas Electronics Corporation

CubeSuite+ V1.03.00

Renesas Electronics Corporation

R20UT2137EJ0100