User's Manual

CcnvCA78K0

C Source Converter

User's Manual

Target Device

RL78 Family

Target Version

V1.00.00 or later

Renesas Electronics
www.renesas.com

Rev.1.00 Apr 2016

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

How to Use This Manual

This manual describes the C source converter (CcnvCA78K0) used for developing application systems for the RL78 family.

Readers

This manual is intended for users who wish to use the CC-RL, which is a C compiler for the RL78 family, to develop application systems.

Purpose

This manual is intended to be used for reference in porting of the development environment of the CA78K0 or the CC78K0, which is a C compiler for 78K0 microcontroller, to the CC-RL.

Organization

This manual can be broadly divided into the following units.

1. GENERAL
2. COMMAND REFERENCE
3. CONVERSION SPECIFICATIONS
4. MESSAGES
5. POINTS FOR CAUTION

How to Read This Manual

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

Data significance:	Higher digits on the left and lower digits on the right
Note:	Footnote for item marked with Note in the text
Caution:	Information requiring particular attention
Remarks:	Supplementary information
Numeric representation:	Decimal ... XXXX
	Hexadecimal ... 0xXXXX

Please refer to the following manuals about CA78K0, CC78K0, and CC-RL. Make sure to refer to the latest versions of these documents. The newest versions of the documents listed may be obtained from the Renesas Electronics Web site.

Compiler	Document Title	Document No.
CA78K0	CubeSuite+ V1.03.00 Integrated Development Environment User's Manual: 78K0 Coding	R20UT2141EJ0100
	CubeSuite+ V1.01.00 Integrated Development Environment User's Manual: 78K0 Build	R20UT0783EJ0100
CC78K0	User's Manual CC78K0 Ver.3.70 C Compiler Language	U17200EJ1V0UM00
	User's Manual CC78K0 Ver.3.70 C Compiler Operation	U17201EJ1V0UM00
CC-RL	CC-RL Compiler User's Manual	R20UT3123EJ0102

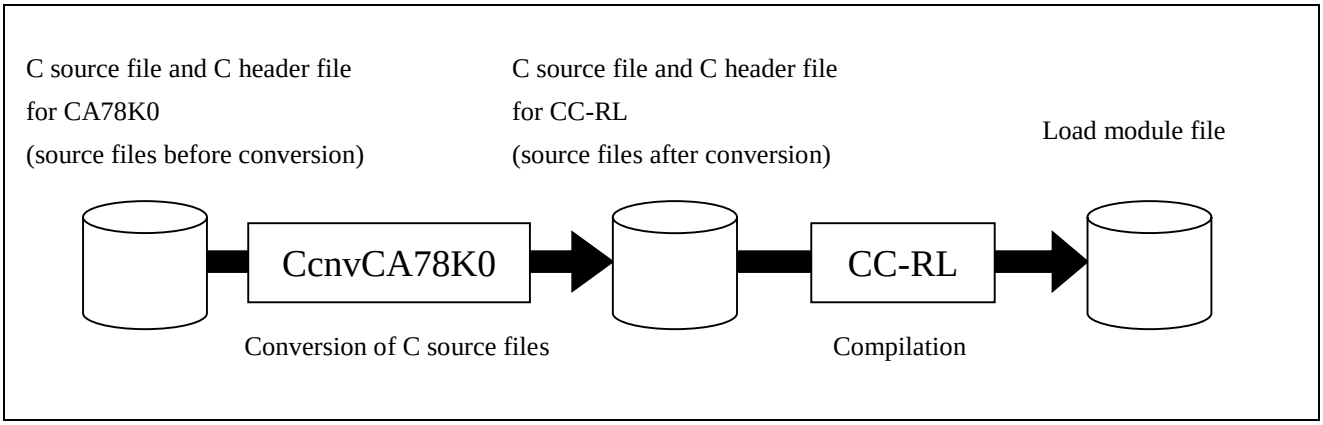
TABLE OF CONTENTS

1. GENERAL

The CcnvCA78K0 is a C source converter that converts C source files created in a development environment using the CA78K0 or the CC78K0 (hereafter CA78K0 and CC78K0 are collectively referred to as CA78K0) which is a C compiler for 78K0 microcontroller into C source files for the CC-RL which is a C compiler for the RL78 family. The extended functions for the CA78K0 written in C source files are converted so that they can be handled by the CC-RL.

CcnvCA78K0 supports the porting of C source files from the CA78K0 compiler to CC-RL.
Since we do not guarantee the correct operation of programs converted by CcnvCA78K0, be sure to check the operation of the C source files after conversion.

Figure 1.1 Overview of CcnvCA78K0



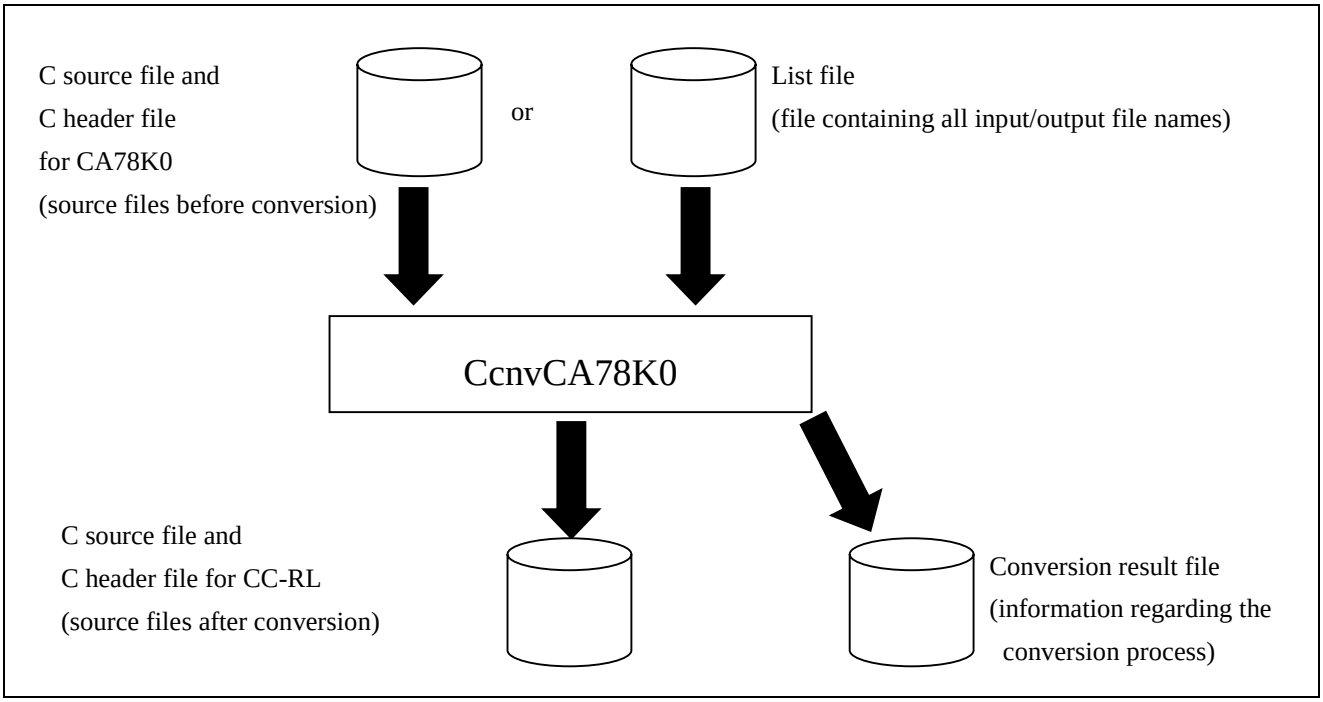
2. COMMAND REFERENCE

This section describes the processing flow in the CcnvCA78K0.

2.1 Overview

The CcnvCA78K0 converts extended language specifications (such as macro names, reserved words, #pragma directives, and extended functions) in C source programs for the CA78K0 into extended language specifications for the CC-RL. Then the CcnvCA78K0 generates C source files for the CC-RL.

Figure 2.1 Processing Flow in CcnvCA78K0



2.2 I/O Files

The I/O files of the CcnvCA78K0 are shown below.

Table 2.1 I/O Files

File Type	I/O	Extension	Description
C source file Header file	I/O	(Input) .c .h (Output) free	A C source file or C header file for the CA78K0 is input and the converted C source file or C header file for the CC-RL is output. The version information of the CcnvCA78K0 is inserted at the beginning of the converted file as a comment and the former description of the converted code is left as a comment. The extension of the input file is fixed. If a file with another extension is specified, the input file is directly output without its contents being converted. The converted file can be specified with the -o option or -l option. If a converted file is re-input, the file is directly output without being converted, and the fact that the file was already converted is notified.
List file	I	free	Text file which includes the input file names and output file names. Specifying the list file with the -l option enables multiple source files to be converted collectively. For the format of the list file, see "-l option".
Conversion result file	O	free	Messages in the conversion result that is output to the standard output file can be output to a file specified by the -r option. For details on the messages, see "MESSAGES".

Examples of an input file and an output file are shown below. For details on conversion specifications, see "CONVERSION SPECIFICATIONS".

(Input file: input.c)

```
#pragma sfr

char c;

void main(void)

{

    c = P0;

}
```

(Output file: output.c)

```
/* CA78K0 C Source Converter Vx.xx.xx.xx [dd Mmm yyyy] */
/************************************************************************
 DISCLAIMER
 This software is supplied by Renesas Electronics Corporation and is only
 intended for use with Renesas products. No other uses are authorized. This
 software is owned by Renesas Electronics Corporation and is protected under
 all applicable laws, including copyright laws.
 THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES REGARDING
 THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT
 LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
 AND NON-INFRINGEMENT. ALL SUCH WARRANTIES ARE EXPRESSLY DISCLAIMED.
 TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
 ELECTRONICS CORPORATION NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
 FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES FOR
 ANY REASON RELATED TO THIS SOFTWARE, EVEN IF RENESAS OR ITS AFFILIATES HAVE
 BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 Renesas reserves the right, without notice, to make changes to this software
 and to discontinue the availability of this software. By using this software,
 you agree to the additional terms and conditions found by accessing the
 following link:
 http://www.renesas.com/disclaimer
 Copyright (C) yyyy Renesas Electronics Corporation. All rights reserved.
************************************************************************/

//[CcnvCA78K0]

#include "iodefine.h"


//[CcnvCA78K0] #pragma sfr

char c;

void main(void)

{

   c = P0;

}
```

2.3 Conversion Result

The CcnvCA78K0 outputs the conversion result to the standard output. The output format is as follows.

```
Message

Input file name
    Result
    Number of messages
```

When the -l option is specified, the above output is repeated for the number of files in the list file.

"Message" is output when there is an error or warning. For the output format of a message, see "MESSAGES".
When the -r option is specified, the message is output not to the standard output file but to the specified file.
"Input file name" is the input file specified on the command line or in the list file.
"Result" displays any one of the following.

- When there is converted code

```
Converted successfully.
```

- When there is no converted code

```
Nothing converted.
```

- When a converted file is re-input to CcnvCA78K0

```
Already converted.
```

- When an error has occurred

```
Conversion failed.
```

"Number of messages" indicates how many messages were output by message type.

An example of the conversion result is shown below.

(Input file: input.c)

```
#pragma sfr
char c;
void main(void)
{
    c = P0;
}
```

(Standard output)

```
CA78K0 C Source Converter Vx.xx.xx.xx [dd Mmm yyyy]

input.c(1):M0592123:[Insert]Inserted #include "iodefine.h".
input.c(1):M0592131:[Delete]#pragma sfr was deleted.
input.c(1):M0592146:[Info]The language specification dependent on 78K0.

input.c
      Converted successfully.
      1 deleted, 1 inserted, 0 changed, 1 information
      Total warning(s) : 0
```

2.4 Method for Manipulating

Input on the command line should be made as follows.

CcnvCA78K0[Δ*option*]…[Δ*file*] [Δ*option*]…

file	: File name
option	: Option name
[ ]	: Can be omitted
…	: Pattern in proceeding [ ] can be repeated
{ }	: Select from items delimited by the pipe symbol ("|")
Δ	: One or more spaces

- Any file names supported by Windows are allowed as input file names or file names to be specified for options.
- Input file names and file names to be specified for options can also be specified with an absolute path or relative path. When specifying an input file name or a file name to be specified for an option without the path or with a relative path, the reference point of the path is the current folder.
- When a space is included in an input file name or a file name to be specified for an option (including the path name), enclose the file name including the path name in a pair of double quotation marks (").
- The maximum length of an input file name or a file name to be specified for an option depends on Windows (up to 259 characters).
- An error will occur when more than one input file name is specified. Use the -l option to specify multiple input file names.
- When an input file is specified, it is certainly necessary to specify an output file name. When an input file has been specified on the command line, use the -o option to specify the output file.
- An error will occur if the same option is specified for more than once.

2.5 Options

This section explains CcnvCA78K0 options.

- Uppercase characters and lowercase characters are distinguished for options.
- When a file name is specified as a parameter, it can include the path (absolute path or relative path). When a file name without the path or a relative path is specified, the reference point of the path is the current folder.
- When a parameter includes a space (such as a path name), enclose the parameter in a pair of double quotation marks (").

Table 2.2 Options

Option	Description
-V	This option displays the version information of CcnvCA78K0.
-h	This option displays the descriptions of CcnvCA78K0 options.
-c	This option specifies the Japanese character code.
-l	This option specifies the list file name.
-o	This option specifies the output file name.
-r	This option specifies where the message is to be output.
-A	This option performs conversion with the functions related to the ANSI standard enabled.

-V

This option displays the version information of CcnvCA78K0.

[Specification format]

-V

・Interpretation when omitted
The version information of CcnvCA78K0 is not displayed.

[Detailed description]

- This option outputs the version information of CcnvCA78K0 to the standard error output.
- Conversion is not performed when this option is specified.
- When this option is specified simultaneously with another option, the other option is ignored.

[Example of use]

>CcnvCA78K0 -V

-h

This option displays the descriptions of CcnvCA78K0 options.

[Specification format]

```
-h
```

・Interpretation when omitted

The descriptions of CcnvCA78K0 options are not displayed.

[Detailed description]

- This option outputs the descriptions of CcnvCA78K0 options to the standard error output.
- Conversion is not performed when this option is specified.
- When this option is specified simultaneously with another option, the other option is ignored.
- When this option is specified simultaneously with the -V option, the -V option is given priority.

[Example of use]

```
>CcnvCA78K0    -h
```

-C

This option specifies the Japanese character code.

[Specification format]

```
-c={none | sjis | euc_jp}
```

・Interpretation when omitted
sjis is assumed as the parameter for this option.

[Detailed description]

- This option specifies the character code to be used for comments in the input file.
- An error will occur if the parameter is omitted.
- The parameters that can be specified are shown below. A warning is output and sjis is assumed if any other item is specified. Operation is not guaranteed if the specified character code differs from the character code of the input file.

none	Does not process the Japanese character code.
sjis	SJIS
euc_jp	EUC (Japanese)

[Example of use]

```
>CcnvCA78K0    input.c    -c=euc_jp    -o=output.c
```

-l

This option specifies the list file name.

[Specification format]

-l=*file*

・Interpretation when omitted
The file specified on the command line is converted.

[Detailed description]

- This option is to be specified when simultaneously converting multiple files.
- An error will occur if the specified list file does not exist.
- When this option is specified, a warning is output for the file name specified on the command line and it is ignored.
- When this option is specified simultaneously with the -o option, a warning is output and the -o option is ignored.
- An error will occur if the parameter is omitted.
- The format of the list file is as follows.

```
[-c={none | sjis | euc_jp}] [-A] input-file-name output-file-name
[-c={none | sjis | euc_jp}] [-A] input-file-name output-file-name
(Omitted from here)
```

[ ] : Can be omitted
{ } : Select from items delimited by the pipe symbol ("|")

- The -c option, -A option, input file name, and output file name are to be specified in this order in one line.
- The -c option and -A option can be omitted. The input and output file names cannot be omitted.
- The input and output file names that can be written are the same as those specifiable on the command line.
- When a space is included in a file name, enclose the file name in a pair of double quotation marks (").
- If the -c option specification on the command line differs from that in the list file, a warning is output and the list file specification is given priority.
- If the output file already exists, it will be overwritten and no warning is output.
- An error will occur if the output file name matches the input file name or the file name specified by the -r option.
- For the list file, only UTF-8N (without BOM) is acceptable for the Japanese character code and only CR+LF is acceptable for the new line code.

[Example of use]

```
>CcnvCA78K0    -l=listfile.txt
```

- Contents of list file (listfile.txt)

```
-c=sjis  input\file1.c  output\file1.c

-c=sjis  input\file2.c  output\file2.c

-c=sjis  input\file.h  output\file.h
```

-o

This option specifies the output file name.

[Specification format]

-o=*file*

・Interpretation when omitted

This option cannot be omitted except for when the -V, -h, or -l option is specified. An error will occur if this option is omitted.

[Detailed description]

- This option specifies the output file name after conversion.
- If the specified file already exists, it will be overwritten and no warning is output.
- An error will occur if the output file name matches the input file name or the file name specified by the -r option.
- When this option is specified simultaneously with the -l option, a warning is output and this -o option is ignored.
- An error will occur if the parameter is omitted.

[Example of use]

```
>CcnvCA78K0    input.c    -o=output.c
```

-r

This option outputs messages to the specified file.

[Specification format]

-r=*file*

・Interpretation when omitted
Messages are output to the standard output file.

[Detailed description]

- This option outputs messages to the specified file.
- If the specified file already exists, it will be overwritten and no warning is output.
- An error will occur if the specified file name matches the input or output file name of the C source file or C header file.
- An error will occur if the parameter is omitted.

[Example of use]

```
>CcnvCA78K0    input.c    -o=output.c    -r=input.txt
```

-A

This option performs conversion with the -za option (which is an ANSI-compliant option of CA78K0) enabled.

[Specification format]

```
-A
```

・Interpretation when omitted
Conversion is performed with the -za option disabled.

[Detailed description]

・ When this option is specified, the following words are not regarded as keywords and they will not be converted.

```
callt,  sreg,  boolean,  bit
```

・ Specify this option if the -za option is used in the CA78K0 development environment before conversion.

[Example of use]

```
>CcnvCA78K0  input.c  -o=output.c  -A
```

3. CONVERSION SPECIFICATIONS

This section shows the conversion specifications of the CcnvCA78K0.

- Correct operation is not guaranteed when a C source program that is syntactically incorrect for the CA78K0 is input.
- The contents included in comments and strings are not converted.
- Nesting of comments is not supported. A nested comment text is not recognized normally and the range of the comment is invalid. Confirm that there are no nested comments before conversion.
- When a keyword that is supposed to be converted cannot be found as a keyword due to some reasons, such as it being generated by a ## operator, the keyword cannot be converted. If the C source program is directly compiled by the CC-RL, a compile error will occur. Confirm that there is no #define, typedef, or ## operator for a keyword to be converted.
- In the CC-RL, an area where to locate memory should be specified with the __near/__far keyword. Since the default operation when the __near/__far keyword is omitted varies according to the memory model specified in the CC-RL, the pointer type may not match after conversion. For a C source program after conversion, specify a small model in the CC-RL. For a microcontroller with the 78K0 memory bank facility, specify the __far keyword for functions in bank 2 and subsequent banks to make those functions have the far attribute.
- Included files in a C source program are not converted. They have to be converted separately.

The following extended language specifications are converted.

- Macro Names
- Reserved Words
- Bit Access
- #pragma section
- ASM Statements
- Interrupt Handler
- Interrupt Handler for RTOS
- Task Function for RTOS
- Absolute Address Allocation Specification
- Intrinsic Functions
- Other #pragma Directives
- Standard Library Functions

3.1 Macro Names

The macros supported in the CA78K0 are converted as follows. If there is no corresponding macro in the CC-RL, the CcnvCA78K0 outputs a message. The CPU macro is not converted and no message is output.

Table 3.1 Conversion of Macro Names

CA78K0 Macro Name	After Conversion	Remarks
__LINE__	Not converted	Can be used in the CC-RL without any change.
__FILE__	Not converted	Can be used in the CC-RL without any change.
__DATE__	Not converted	Can be used in the CC-RL without any change.
__TIME__	Not converted	Can be used in the CC-RL without any change.
__STDC__	Not converted	Can be used in the CC-RL without any change.
__K0__	Not converted	A message is output. Handled as a user-defined macro in the CC-RL.
__STATIC_MODEL__	Not converted	A message is output. Handled as a user-defined macro in the CC-RL.
__CHAR__UNSIGNED__	__UCHAR	
__CA78K0__	Not converted	A message is output. Handled as a user-defined macro in the CC-RL.
CPU macro	Not converted	A message is not output. Handled as a user-defined macro in the CC-RL.

3.2 Reserved Words

The conversion specifications for reserved words are shown here.

Table 3.2 Conversion of Reserved Words

CA78K0 Reserved Word	After Conversion	Remarks
__callt	Not converted	Can be used in the CC-RL without any change.
callt	__callt	Converted only when the -A option is invalid.
__callf	Deleted	
callf	Deleted	Deleted only when the -A option is invalid.
__sreg	__saddr	Always converted.
sreg	__saddr	Converted only when the -A option is invalid.
noauto	Deleted	Deleted only when the -A option is invalid.
__leaf	Deleted	
norec	Deleted	Deleted only when the -A option is invalid.
__boolean	_Bool	When the -ansi option is specified in the CC-RL, change the _Bool type to the char type.
boolean	_Bool	Converted only when the -A option is invalid.
bit	_Bool	Converted only when the -A option is invalid.
__interrupt	#pragma interrupt	For details, see "Interrupt Handler".
__interrupt_brk	#pragma interrupt_brk	For details, see "Interrupt Handler".
__asm	#pragma inline_asm	For details, see "ASM Statements".
__rtos_interrupt	#pragma rtos_interrupt	For details, see "Interrupt Handler for RTOS".
__directmap	#pragma address	For details, see "Absolute Address Allocation Specification".
__pascal	Deleted	
__flash	Deleted	
__flashf	Deleted	
__temp	Deleted	
__mxcall	Deleted	
Bank function (__BANK0, …)	Not converted	A message is not output. Handled as a user-defined function in the CC-RL.

3.3 Bit Access

The CC-RL does not support bit access (specifying the bit position after a period for an SFR or the saddr variable) of the CA78K0. In the CcnvCA78K0, bit access for SFRs and the saddr variable are replaced with a type declaration of a bit field and a macro.

- The type declaration and macro are output at the beginning of the file and changed to a macro call at an access point.
- In bit access, a bit field of 8 or 16 bits is created according to the bit position. If the bit position includes 8 to 15, a bit field with b8 to b15 added is separately created for 16 bits.

[Examples]

- Bit position is only 0 to 7

Before conversion

```
void func(void)
{
  i = var.3;
  var.5 = 0;
}
```

After conversion

```
#ifndef __BIT8
typedef struct {
  unsigned int b0:1;
  unsigned int b1:1;
  unsigned int b2:1;
  unsigned int b3:1;
  unsigned int b4:1;
  unsigned int b5:1;
  unsigned int b6:1;
  unsigned int b7:1;
} __Bits8;
#define __BIT8(name,bit) (((volatile __near __Bits8*)&name)->b##bit)
#endif

void func(void)
{
  i = __BIT8(var,3);
  __BIT8(var,5) = 0;
}
```

・ Bit position includes 8 to 15

Before conversion

```
i = var2.10;
var2.12 = 0;
```

After conversion

```
#ifndef __BIT16
typedef struct {
  unsigned int b0:1;
  unsigned int b1:1;
  unsigned int b2:1;
  unsigned int b3:1;
  unsigned int b4:1;
  unsigned int b5:1;
  unsigned int b6:1;
  unsigned int b7:1;
  unsigned int b8:1;
  unsigned int b9:1;
  unsigned int b10:1;
  unsigned int b11:1;
  unsigned int b12:1;
  unsigned int b13:1;
  unsigned int b14:1;
  unsigned int b15:1;
} __Bits16;
#define __BIT16(name,bit) (((volatile __near __Bits16*)&name)->b##bit)
#endif

void func(void)
{
  i = __BIT16(var2,10);
  __BIT16(var2,12) = 0;
}
```

3.4 #pragma section

#pragma section requires the section name to be converted because the section names differ between the CA78K0 and CC-RL. However, some sections cannot be converted because there are no corresponding sections on the CC-RL side. Though conversion is possible, some sections have slightly different facilities. The CcnvCA78K0 outputs a message to the standard error output upon conversion of some sections. For details, see "Correspondence Table of Section Names".

The format of the CA78K0 is as follows.

```
#pragma section section-name changed-section-name [AT-start-address]
```

The format of the CC-RL is as follows.

```
#pragma section [{text | const | data | bss}] [changed-section-name]
```

- Since the CC-RL does not have a facility equivalent to "AT-start-address", if there is "AT-start-address", the CcnvCA78K0 deletes it and outputs a message. Use the -start option to specify the location of the section in the CC-RL. For details on the -start option, see the user's manual of the CC-RL.
- "changed-section-name" is directly output without being converted. If a character unusable in the CC-RL (e.g., ?) is used in the changed section name, a compile error will occur in the CC-RL. Change the string after conversion.
- In #pragma section of the CC-RL, the section name is "changed section name + _n" or "changed section name + _f", and the section name for the saddr area is "changed section name + _s". For details, see the user's manual of the CC-RL.
- If conversion is not possible because there is no corresponding section in the CC-RL, the CcnvCA78K0 outputs a message and does not perform conversion. Then the CC-RL outputs a message and ignores the #pragma directive. Modify the C source program in accordance with the Correspondence Table of Section Names described later.

[Examples]

Pattern 1 (Replaced successfully)	Before conversion	#pragma section @@CODE MY_CODE
	After conversion	#pragma section text MY_CODE
Pattern 2 (Deletion of AT)	Before conversion	#pragma section @@CODE MY_CODE AT 0x2000
	After conversion	#pragma section text MY_CODE
Pattern 3 (Compile error after replacement)	Before conversion	#pragma section @@CODE ??CODE AT 0x2000
	After conversion	#pragma section text ??CODE
	Corrective action	Though conversion is performed, an error will occur at compilation. Change the section name.
Pattern 4 (Replacement is not possible)	Before conversion	#pragma section @@CALF MY_BASE
	After conversion	#pragma section @@CALF MY_BASE
	Corrective action	Since there is no corresponding section in the CC-RL, the program is output without being converted. Correct the program according to the Correspondence Table of Section Names.

Table 3.3 Correspondence Table of Section Names

CA78K0 Section Name	Description	CC-RL Section Type	CcnvCA78K0 Operation Corrective Action after Conversion
@@CODE @ECODE	Segment for code portion	text	The section is changed to the corresponding section type.
			No action is required. The section name in the CC-RL is "changed section name + _n" or "changed section name + _f".
@@LCODE @LECODE	Segment for library code	text	Conversion is not performed.
			Delete #pragma. Specify the location of the library in the CC-RL with the link option -ROm.
@@CNST @ECNST	Segment for ROM data	const	The section is changed to the corresponding section type.
			No action is required. The section name in the CC-RL is "changed section name + _n".
@@R_INIT @ER_INIT	Segment for initialized data	data	The section is changed to the corresponding section type.
			No action is required. The section name in the CC-RL is "changed section name + _n".
@@R_INIS @ER_INIS	Segment for initialized data (sreg variable)	data	The section is changed to the corresponding section type.
			No action is required. The section name in the CC-RL is "changed section name + _s".
@@CALF	Segment for callf function	None	A message is output and conversion is not performed.
			Delete #pragma. There is no corresponding facility in the CC-RL. Processing needs to be reviewed.
@@CALT	Segment for callt function table	None	A message is output and conversion is not performed.
			Delete #pragma. The section name cannot be changed in the CC-RL.
@@VECTnn @EVECTnn	Segment for vector table	None	Conversion is not performed.
			Delete #pragma. The section name cannot be changed in the CC-RL.

CA78K0 Section Name	Description	CC-RL Section Type	CcnvCA78K0 Operation Corrective Action after Conversion
@EXTxx	Segment for flash area branch table	None	Conversion is not performed.
			Delete #pragma. There is no corresponding facility in the CC-RL. Processing needs to be reviewed.
@@INIT @EINIT	Segment for data area (initialized)	None	A message is output and conversion is not performed.
			Delete #pragma. Specify the section for mapping ROM to RAM with the link option -ROm.
@@INIS @EINIS	Segment for data area (sreg variable, initialized)	None	A message is output and conversion is not performed.
			Delete #pragma. Specify the section for mapping ROM to RAM with the link option -ROm.
@@DATA @EDATA	Segment for data area (uninitialized)	bss	The section is changed to the corresponding section type.
			No action is required. The section name in the CC-RL is "changed section name + _n".
@@DATS @EDATS	Segment for data area (sreg variable, uninitialized)	bss	The section is changed to the corresponding section type.
			No action is required. The section name in the CC-RL is "changed section name + _s".
@@BITS @EBITS	Segment for boolean type and bit type variables	None	A message is output and conversion is not performed.
			Delete #pragma. The section is allocated to the same section as other data as the _Bool type in the CC-RL.
@@BANK0, …, @@BANK15	Segment for bank function	None	A message is output and conversion is not performed.
			Delete #pragma. There is no corresponding facility in the CC-RL. Processing needs to be reviewed.

3.5 ASM Statements

The __asm() function or #asm-#endasm is used to write assembly-language code within functions for the CA78K0, whereas inline expansion is performed for the assembly-language functions declared in #pragma inline_asm for the CC-RL. The CcnvCA78K0 creates the __asm() function or the inline_asm function that executes assembly instructions in the range between #asm and #endasm at the beginning of the file and converts the program so that this function is called at the position where an assembly instruction is written.

The format of the CA78K0 is as follows.

```
#asm
   :       /* assembly-language code */
#endasm
```

```
__asm("assembly-language code");
```

The format of the CC-RL is as follows.

```
#pragma inline_asm [(] function-name [, … ] [)]
function-declaration {
   :       /* assembly-language code */
}
```

- Since the instruction set or specifications of instructions are different between the 78K0 and RL78, the assembly-language code has to be manually modified. A message is output at conversion.
- A tab is appended as an indent to the assembly-language code within the inline_asm function.
- The function name to be created should be in the range between __inline_asm_func_00000 and __inline_asm_func_99999, and an error will occur if the number of functions exceeds 100,000.
- If a label is in the range between #asm and #endasm or in the __asm function, the CcnvCA78K0 outputs a message. If a label is written in a function for which #pragma inline_asm is specified in the CC-RL, an error will occur at compilation. Therefore, if a label is in #asm-#endasm or the __asm function, the CcnvCA78K0 outputs a message. A label written in the assembly language needs to be changed to a local label to avoid a compile error. For details, see the user's manual of the CC-RL.
- If double quotation marks (") are included in the target to be converted by the #define macro as shown in the example below, the inline_asm function cannot be generated from the __asm( ) function. In such a case, the CcnvCA78K0 outputs a message. The input file is directly output without its contents being converted. Perform conversion after expanding the macro in advance.

  Example) #define MAC "nop"
  __asm(MAC);
- If control characters like '\n' or '\t' are included in a string in __asm( ), an assembly error will occur after conversion. Perform conversion after deleting the control characters in advance.
- If a C-language comment ("/*") is included in the assembly-language comments (";") in the range between #asm and #endasm, the range of the comment is invalid. Perform conversion after deleting the comments in advance.

[Examples]

Pattern 1	Before conversion	`void func()`
		`{`
		`  __asm("nop");`
		`}`

	After conversion	`#pragma inline_asm __inline_asm_func_00000`
		`static void __inline_asm_func_00000(void)`
		`{`
		`  nop`
		`}`
		
		`void func()`
		`{`
		`  __inline_asm_func_00000();`
		`}`

Pattern 2	Before conversion	`void func(void)`
		`{`
		`  #asm`
		`  nop`
		`  #endasm`
		`}`

	After conversion	`#pragma inline_asm __inline_asm_func_00001`
		`static void __inline_asm_func_00001(void)`
		`{`
		`  nop`
		`}`
		
		`void func()`
		`{`
		`  __inline_asm_func_00001();`
		`}`

Pattern 3	Before conversion	`#define ASM_NOP __asm("nop");`

	After conversion	`#pragma inline_asm __inline_asm_func_00002`
		`static void __inline_asm_func_00002(void)`
		`{`
		`  nop`
		`}`
		`#define ASM_NOP __inline_asm_func_00002();`

Pattern 4
(Error after conversion)
	Before conversion	void func()
		{
		__asm("\tnop");
		}

	After conversion	#pragma inline_asm __inline_asm_func_00003
		static void __inline_asm_func_00003(void)
		{
		\tnop
		}
		
		void func()
		{
		__inline_asm_func_00003();
		}

3.6 Interrupt Handler

#pragma interrupt/vect and the __interrupt and __interrupt_brk keywords of the CA78K0 are converted into #pragma interrupt/interrupt_brk of the CC-RL.

The format of an interrupt function of the CA78K0 is as follows.

```
<Normal model>
#pragma interrupt(vect) interrupt-request-name function-name function-name
                [Stack-change-specification]
                [{Stack-usage-specification | No-change-specification | Register-bank-specification}]
<Static model>
#pragma interrupt(vect) interrupt-request-name function-name function-name
                [{Shared-area-save/restore-specification | Save/restore-target}]
                [{Stack-usage-specification | No-change-specification | Register-bank-specification}]
```

or

```
__interrupt void func() { processing }
__interrupt_brk void func() { processing }
```

The format of an interrupt function of the CC-RL is as follows.

```
#pragma interrupt [(] function-name [([vect=address][,bank=register-bank][,enable={true|false}])][)]
function-declaration
#pragma interrupt_brk [(] function-name [([bank=register-bank][,enable={true|false}])][)]
function-declaration
```

- When the interrupt request name exists, #include "iodefine.h" is output. A message is output because the interrupt request name may not be appropriate due to the device being changed.
- __interrupt is converted into #pragma interrupt and __interrupt_brk is converted into #pragma interrupt_brk.
- When the interrupt request name is BRK_I, it is converted into #pragma interrupt_brk.
- "interrupt-request-name" is converted into "vect=address" as a macro that indicates the address. The macro value is defined by iodefine.h.
- "Register-bank-specification" is converted into "bank=register-bank".
- Since "Stack-change-specification", "Stack-usage-specification", "No-change-specification", "Shared-area-save/restore-specification", and "Save/restore-target" do not exist in the CC-RL, the CcnvCA78K0 outputs a message and deletes them.
- When a macro or typedef is used in declaration or definition of an interrupt function using the __interrupt or __interrupt_brk keyword, the function name may be interpreted erroneously. Perform conversion after expanding the macro or typedef in advance.
- If there is a #pragma directive and a description of an interrupt function using a keyword for the same function, converting both of them into #pragma directives sometimes generates duplicate #pragma directives after conversion and a compile error will occur. In this case, delete the duplicate description.
- When omitting parameters of a function declaration in which the __interrupt or __interrupt_brk keyword is specified, a compile error will occur in the CC-RL. The void type has to be written as the parameter type.

[Examples]

Pattern 1	Before conversion	#pragma vect INTP0 func sp=buff+10 rb1 void func(void) { }
	After conversion	#pragma interrupt func(vect=INTP0, bank=RB1) void func(void) { }
Pattern 2	Before conversion	#pragma interrupt INTP0 func leafwork1 rb1 void func(void) { }
	After conversion	#pragma interrupt func(vect=INTP0, bank=RB1) void func(void) { }
Pattern 3	Before conversion	__interrupt void func(void) { }
	After conversion	#pragma interrupt func void func(void) { }
Pattern 4	Before conversion	#pragma interrupt BRK_I func void func(void) { }
	After conversion	#pragma interrupt_brk func void func(void) { }
Pattern 5	Before conversion	__interrupt void func1(void), func2(void);

	After conversion	#pragma interrupt func1
		void func1(void);
		#pragma interrupt func2
		void func2(void);

Pattern 6	Before conversion	#pragma interrupt INTP0 func __interrupt func(void);
	After conversion	#pragma interrupt func(vect=INTP0) void func(void); #pragma interrupt func void func(void);
	Corrective action	Duplicate #pragma directives will cause an error in the CC-RL. Delete one of the #pragma directives.
Pattern 7	Before conversion	typedef void func_t(void); __interrupt func_t f1;
	After conversion	typedef void func_t(void); __interrupt func_t f1;
	Corrective action	A compile error will occur in the CC-RL. Expand typedef or the macro in advance.

3.7 Interrupt Handler for RTOS

#pragma rtos_interrupt and the __rtos_interrupt keyword of the CA78K0 are converted into #pragma rtos_interrupt of the CC-RL.

The format of the CA78K0 is as follows.

```
#pragma rtos_interrupt [interrupt-request-name function-name [Stack-change-specification]]
```

or

```
__rtos_interrupt function-declaration
```

The format of the CC-RL is as follows.

```
#pragma rtos_interrupt [(] function-name [(vect=address)][)]
function-declaration
```

- When the interrupt request name exists, #include "iodefine.h" is output. A message is output because the interrupt request name may not be appropriate due to the device being changed.
- __rtos_interrupt is converted into #pragma rtos_interrupt.
- "interrupt-request-name" is converted into "vect=address" as a macro that indicates the address. The macro value is defined by iodefine.h.
- Function names can be omitted in the format of the CA78K0 and so the CA78K0 has a facility that prevents the user from defining ret_int and ret_wup which are used by the RTOS interrupt handler. Since the same facility is not available in the CC-RL, if the interrupt request name and function name are omitted, the CcnvCA78K0 outputs a message and comments out the #pragma directive.
- Since "Stack-change-specification" is not available in the CC-RL, it is deleted and a message is output.
- When a macro or typedef is used in declaration or definition of an interrupt function using the __rtos_interrupt keyword, the function name may be interpreted erroneously. Perform conversion after expanding the macro or typedef in advance.
- If there is a #pragma directive and a description of an interrupt function by a keyword for the same function, converting both of them into #pragma directives sometimes generates duplicate #pragma directives after conversion and a compile error will occur. In this case, delete the duplicate description.
- When omitting parameters of a function declaration in which the __rtos_interrupt keyword is specified, a compile error will occur in the CC-RL. The void type has to be written as the parameter type.

[Examples]

Pattern 1	Before conversion	#pragma rtos_interrupt INTP0 func void func(void) { }
	After conversion	#pragma rtos_interrupt func (vect=INTP0) void func(void) { }
Pattern 2	Before conversion	#pragma rtos_interrupt INTP0 func sp=buff+10 void func(void) { }
	After conversion	#pragma rtos_interrupt func (vect=INTP0) void func(void) { }
Pattern 3	Before conversion	__rtos_interrupt void func(void) { }
	After conversion	#pragma rtos_interrupt func void func(void) { }
Pattern 4	Before conversion	#pragma rtos_interrupt
	After conversion	// #pragma rtos_interrupt
Pattern 5	Before conversion	__rtos_interrupt void func1(void), func2(void);

	After conversion	#pragma rtos_interrupt func1
		void func1(void);
		#pragma rtos_interrupt func2
		void func2(void);

Pattern 6	Before conversion	#pragma rtos_interrupt INTP0 func __rtos_interrupt func(void);
	After conversion	#pragma rtos_interrupt func(vect=INTP0) void func(void); #pragma rtos_interrupt func void func(void);
	Corrective action	Duplicate #pragma directives will cause an error in the CC-RL. Delete one of the #pragma directives.
Pattern 7	Before conversion	typedef void func_t(void); __rtos_interrupt func_t f1;
	After conversion	typedef void func_t(void); __rtos_interrupt func_t f1;
	Corrective action	A compile error will occur in the CC-RL. Expand typedef or the macro in advance.

3.8 Task Function for RTOS

The format of the task functions for RTOS is almost the same in the CA78K0 and CC-RL.

The format of the CA78K0 is as follows.

```
#pragma rtos_task [task-function-name]
```

The format of the CC-RL is as follows.

```
#pragma rtos_task [(] task-function-name [, … ][)]
function-declaration
```

- Task function names can be omitted in the format of the CA78K0 and so the CA78K0 has a facility that prevents the user from defining ext_tsk which is used by the task functions for RTOS. Since the same facility is not available in the CC-RL, if the task function name is omitted, the CcnvCA78K0 outputs a message and comments out the #pragma directive.

[Examples]

Pattern 1	Before conversion	#pragma rtos_task task1
	After conversion	#pragma rtos_task task1
Pattern 2	Before conversion	#pragma rtos_task
	After conversion	// #pragma rtos_task

3.9 Absolute Address Allocation Specification

The destination is specified using the __directmap keyword in the CA78K0, whereas #pragma address is written immediately before the variable declaration in the CC-RL.

The format of the CA78K0 is as follows.

```
__directmap [__sreg] [static] type-name variable-name = location-address;
```

The format of the CC-RL is as follows.

```
#pragma address variable-name = location-address
variable-declaration
```

- Since the memory map is different between the 78K0 and RL78, a message is output.
- The CcnvCA78K0 deletes the __directmap keyword and adds #pragma address just before the variable declaration. The address specification is deleted from the variable declaration and execution moves to the address specification of #pragma address.
- When a macro or function pointer is used in a description using the __directmap keyword, the function name may be interpreted erroneously. Perform conversion after expanding the macro in advance. The location specification of the function pointer has to be modified manually.
- If different variables are assigned to the same address with __directmap, a compile error will occur in the CC-RL after conversion. Care is required because the CcnvCA78K0 does not check whether different variables are being assigned to the same address.

[Examples]

Pattern 1	Before conversion	__directmap int i = 0xfe00;
	After conversion	#pragma address i=0xfe00 int i;
Pattern 2	Before conversion	__directmap int* i = 0xfe00;
	After conversion	#pragma address i=0xfe00 int* i;
Pattern 3	Before conversion	__directmap int i = 0xfe00, j=0xfe10;
	After conversion	#pragma address i=0xfe00 #pragma address j=0xfe10 int i,j;
Pattern 4	Before conversion	__directmap struct x { char a ; char b ; } xx = { 0xfe30 } ;
	After conversion	#pragma address xx=0xfe30 struct x { char a ; char b ; } xx;
Pattern 5	Before conversion	#define MY_MACRO1 (int i = 0xfe00) __directmap MY_MACRO1;
	After conversion	#define MY_MACRO1 (int i = 0xfe00) __directmap MY_MACRO1;
	Corrective action	Perform conversion after expending the macro.
Pattern 6	Before conversion	__directmap void (*fp[])(void) = 0x1234;
	After conversion	#pragma address void=0x1234 void (*fp[])(void);
	Corrective action	Manually write #pragma address for the CC-RL.

3.10 Intrinsic Functions

Intrinsic functions were validated by #pragma directives in the CA78K0, whereas intrinsic functions can always be used in the CC-RL. If there is an intrinsic function of the CC-RL that corresponds to an intrinsic function of the CA78K0, the CcnvCA78K0 deletes the relevant #pragma directive in the C source program and changes the code where the intrinsic function is called.

- If there is no relevant #pragma directive, the intrinsic function is determined to be invalid and it will not be converted.
- The CcnvCA78K0 deletes #pragma directives for the intrinsic functions that are not supported in the CC-RL and outputs a message. The code where the intrinsic functions are called will not be converted.

Table 3.4 Conversion of Intrinsic Functions

CA78K0 Intrinsic Function	After Conversion	Remarks
#pragma DI DI	Deleted __DI	
#pragma EI EI	Deleted __EI	
#pragma HALT HALT	Deleted __halt	
#pragma STOP STOP	Deleted __stop	
#pragma BRK BRK	Deleted __brk	
#pragma NOP NOP	Deleted __nop	
#pragma rot rolb rorb rolw rorw	Deleted __rolb __rorb __rolw __rorw	
#pragma mul mulu	Deleted __mulu	
#pragma div divuw moduw	Deleted __divui __remui	
#pragma bcd adbcdb, sbbcdb, adbcdbe, sbbcdbe, adbcdw, sbbcdw, adbcdwe, sbbcdwe, bcdtob, btobcde, bcdtow, wtobcd, btobcd	Deleted Not any one of them is converted.	Not supported in the CC-RL.

CA78K0 Intrinsic Function	After Conversion	Remarks
#pragma opc __OPC	Deleted Not converted	Not supported in the CC-RL.
#pragma realregister __geta, __seta, __getax, __setax, __getcy, __setcy, __set1cy, __clr1cy, __not1cy, __inca, __deca, __rora, __rorca, __rola, __rolca, __shla, __shra, __ashra, __nega, __coma, __absa	Deleted Not any one of them is converted.	Not supported in the CC-RL.
#pragma hromcall __hromcall __hromcalla __setsp	Deleted Not any one of them is converted.	Not supported in the CC-RL.
#pragma access peekb, peekw, pokeb, pokew	Deleted Not any one of them is converted.	Not supported in the CC-RL.

3.11 Other #pragma Directives

Conversion specifications for other #pragma directives are shown here.

Table 3.5 Conversion of Other #pragma Directives

CA78K0 #pragma Directive	After conversion	Remarks
#pragma sfr	#include "iodefine.h"	iodefine.h is provided by the integrated development environment. Access to an SFR needs to be reviewed because the device is changed. A message is output.
#pragma name	Deleted	Not supported in the CC-RL.
#pragma ext_table	Deleted	Not supported in the CC-RL.
#pragma ext_func	Deleted	Not supported in the CC-RL.
#pragma inline	Deleted	Not supported in the CC-RL. #pragma inline of the CC-RL is a different facility.
#pragma PC	Deleted	Not supported in the CC-RL.

3.12 Standard Library Functions

Among the standard library functions of the CA78K0, calls of va_starttop, va_start_banked, and va_starttop_banked are converted into standard library functions of the CC-RL. Normal standard library functions are not converted because the same functions are available.

- Do not use the CcnvCA78K0 to convert the header file of the standard libraries for the CA78K0 and make the CC-RL handle the converted header file. Use the header file of the standard libraries for the CC-RL.
- Since the function name is converted by replacing strings, if a macro name, variable name, tag name, etc. of the same name exists, they will also be replaced.
- Since __near/__far is specified for the type of parameters or return values in standard libraries of the CC-RL, the type of parameters or return values may not match after conversion. Manually modify the code after confirming the user's manual of the CC-RL.

Table 3.6 Conversion of Standard Library Functions

Function Name of CA78K0	After Conversion	Remarks
toup _toupper	Not converted	Handled as a user function in the CC-RL. Use the toupper function.
tolow _tolower	Not converted	Handled as a user function in the CC-RL. Use the tolower function.
va_starttop va_start_banked va_starttop_banked	va_start	
atexit	Not converted	atexit is not supported in the CC-RL. Handled as a user function in the CC-RL.
brk	Not converted	Handled as a user function in the CC-RL.
sbrk	Not converted	Handled as a user function in the CC-RL.
itoa	Not converted	Handled as a user function in the CC-RL.
ltoa	Not converted	Handled as a user function in the CC-RL.
ultoa	Not converted	Handled as a user function in the CC-RL.
strbrk	Not converted	Handled as a user function in the CC-RL.
strsbrk	Not converted	Handled as a user function in the CC-RL.
stritoa	Not converted	Handled as a user function in the CC-RL.
strltoa	Not converted	Handled as a user function in the CC-RL.
strultoa	Not converted	Handled as a user function in the CC-RL.
strcoll	Not converted	strcoll is not supported in the CC-RL. Handled as a user function in the CC-RL.
strxfrm	Not converted	strxfrm is not supported in the CC-RL. Handled as a user function in the CC-RL.
matherr	Not converted	Handled as a user function in the CC-RL.
__assertfail	Not converted	Handled as a user function in the CC-RL. The assert macro can be used in the CC-RL without any change.

3.13 Difference from Conversion Specifications of -convert_cc Option of CC-RL

The CC-RL does not have any option for compiling the source code of the CA78K0 at compilation.
The table below shows the differences regarding the extended functions whose operations differ between when the "-convert_cc=ca78k0r" option for compiling the source code of the CA78K0R has been specified in the CC-RL and when conversion by the CcnvCA78K0 has been performed.

Table 3.7 Different Operation from -convert_cc=ca78k0r Option of CC-RL

CA78K0 Extended Function	Operation when -convert_cc=ca78k0r Option is Used	Conversion by CcnvCA78K0
__boolean	Handled as the _Bool type when the -ansi option is not specified and handled as the char type when the -ansi option is specified.	Always converted to the _Bool type. Since _Bool is not usable when the -ansi option of the CC-RL is specified, manually change the type.
__callf __leaf __pascal __flash __flashf __temp __mxcall	A compile error will occur.	Deleted.
callf noauto norec	A compile error will occur.	Deleted when the -A option is not specified.
__interrupt __interrupt_brk __rtos_interrupt	If a #pragma directive for a function with the same name already exists, the keyword is ignored.	A #pragma directive is added before the function declaration. If a #pragma directive for a function with the same name already exists, there will be duplicate #pragma directives after conversion and a compile error will occur. In such a case, delete the #pragma directive that was converted from the keyword.
__asm( )	Recognized as a normal function call. It needs to be manually modified to the inline_asm function. Review the assembly-language code because the device is changed.	#pragma inline_asm and a function definition are output for each __asm( ). A function call of __asm( ) is converted into a newly generated function call. Review the assembly-language code because the device is changed.
va_start_banked va_starttop_banked	Recognized as a normal function call.	Converted to va_start.
#pragma sfr	Use the -preinclude option of the CC-RL to include iodefine.h. Review access to an SFR because the device is changed.	#include "iodefine.h" is inserted at the beginning of the file. Review access to an SFR because the device is changed.

CA78K0 Extended Function	Operation when -convert_cc=ca78k0r Option is Used	Conversion by CcnvCA78K0
#asm-#endasm	A syntax error will occur. It needs to be manually modified to the inline_asm function. Review the assembly-language code because the device is changed.	#pragma inline_asm and a function definition are output for each #asm-#endasm. #asm-#endasm is converted into a newly generated function call. Review the assembly-language code because the device is changed.
#pragma interrupt #pragma vect #pragma rtos_interrupt	If there is an interrupt request name, use the -preinclude option of the CC-RL to include iodefine.h. Review the interrupt request name because the device is changed.	#include "iodefine.h" is inserted at the beginning of the file if there is an interrupt request name. Review the interrupt request name because the device is changed.

4. MESSAGES

This section describes messages that are output by the CC-RL.

4.1 Message Formats

The output formats of messages are as follows.

- When the file name and line number are included
  - Message number type is information

```
file-name (line-number):message-number:[information-type] message
```

The information type is change, insertion, deletion, or information.

  - Message number type is other than information

```
file-name (line-number):message-number:message
```

- When the file name and line number are not included

```
message-number:message
```

The message number is output as a consecutive string consisting of one alphabetic character, 0592, and a three-digit number.

4.2 Message Types

The message types are classified as follows.

Table 4.1 Message Types

Message Type	First Letter	Description
Internal error	C	Processing is aborted. The C source program is not output after conversion.
Fatal Error	E	Processing is aborted. The C source program is not output after conversion.
Warning	W	Processing continues. The C source program is output after conversion.
Information	M	Processing continues. The C source program is output after conversion.

4.3 Information Types

When the message number type is information, the information types are classified as follows.

Table 4.2 Information Types

Information Type	Description
Change	Changes were made in the program so that it can be handled by the CC-RL.
Insert	Additions were made in the program so that it can be handled by the CC-RL.
Delete	Some descriptions were deleted because they are not necessary in the CC-RL.
Info	Conversion may not be sufficient in some cases because of the difference between the CA78K0 and CC-RL specifications. Each case should be confirmed individually.

4.4 Messages

The messages output by the CcnvCA78K0 are as follows.

4.4.1 Internal Errors

Table 4.3 Internal Errors

Number	Message	Description
C0592*nnn*	Internal error	Please contact your vendor or your Renesas Electronics overseas representative.

nnn is a three-digit decimal number.

4.4.2 Fatal Errors

Table 4.4 Fatal Errors

Number	Message	Description
E0592001	Multiple input files are not allowed.	Only one input file can be specified. Use the list file to specify multiple input files.
E0592002	The *option* option cannot have an argument.	An argument was specified for an option that should not have arguments.
E0592003	The *option* option requires an argument.	No argument was specified in an option that requires arguments.
E0592004	The *option* option is specified more than once.	Only one option can be specified at one time.
E0592005	Requires an output file.	The output file corresponding to the input file was not specified.
E0592006	Failed to read an input file *file*.	The folder name or file name may be incorrect. If the next file is specified in the list file, conversion of that file will start.
E0592007	Failed to write a result of conversion file *file*.	The folder name may be incorrect.
E0592008	Failed to write an output file *file*.	The folder name may be incorrect.
E0592009	Failed to read a list file *file*.	The folder name may be incorrect.
E0592010	Syntax errors in list file *file*.	The description of the list file is not correct.
E0592011	File name is corrupted.	There are duplicate file names among the input file, output file, and conversion result output file.
E0592012	Invalid file name.	Either the input file name specified on the command line or an input or output file name specified in the list file has exceeded 260 characters.
E0592013	Invalid argument for the *option* option.	The argument specification is invalid or the specified file name has exceeded 260 characters.
E0592101	Illegal syntax in *string*.	Conversion could not be performed because there was a syntax that is not allowed in the CA78K0. Modify the input file.
E0592102	Can not add inline function for assembly.	The number of inline functions for assembly has exceeded the upper limit. Modify the input file.
E0592103	Failed to delete a temporary file.	Deletion of a temporary file has failed. Delete the temporary file.

4.4.3 Warnings

Table 4.5 Warnings

Number	Message	Description
W0592051	Input file specified on the command line is ignored when the "-l" option is specified.	When the list file is specified, an input file cannot be specified on the command line at the same time. The list file specified by the "-l" option is converted and the input file is ignored.
W0592052	The "-c" option specified on the command line is ignored when it does not match the specification in list file (file).	The "-c" option specification corresponding to the input file "*file*" specified in the list file differs between the list file and command line. Conversion is performed in accordance with the specification in the list file.
W0592053	Invalid *option* option.	An invalid option was specified. Ignore the option.
W0592054	Invalid argument for the *option* option.	The argument specified in the "*option*" option is invalid. If the argument of the "-c" option is invalid, processing is performed with the default specification.
W0592055	Requires an input file.	The list file specified by the "-l" option is missing an input file specification.
W0592151	*String* cannot be changed to syntax of CC-RL.	*string* could not be changed to the CC-RL format. Modify the input file.

4.4.4 Information

Table 4.6 Information

Number	Information Type	Message	Description
M0592111	Change	*String1* was converted into *string2*.	The token was converted.
M0592112	Change	Bit access of I/O register was converted into macro call.	Since the bit access method of SFRs differs between the CA78K0 and CC-RL, the method is changed to make access using a macro.
M0592113	Change	'*String*' has been changed to syntax of CC-RL.	Since the description format differs between the CA78K0 and CC-RL, the description format is changed to that of the CC-RL.

Number	Information Type	Message	Description
M0592121	Insert	Inserted macro definition for bit access of I/O register.	Since the bit access method of SFRs differs between the CA78K0 and CC-RL, the method is changed to make access using a macro.
M0592122	Insert	Inserted #pragma interrupt NO_VECT.	#pragma interrupt without the vector table specification was generated.
M0592123	Insert	Inserted *string*.	A description in accordance with the CC-RL format was added.
M0592124	Insert	Add inline function *string* for assembly.	An inline function for assembly was generated.
M0592125	Insert	Inserted #pragma rtos_interrupt NO_VECT.	#pragma rtos_interrupt without the vector table specification was generated.
M0592131	Delete	*String* was deleted.	The description format is not available in the CC-RL. The description was deleted.
M0592142	Info	The *section* can not be converted. Because there is no matched section.	The section could not be converted because there is no corresponding section in the CC-RL.
M0592143	Info	Delete "AT start address".	This was deleted because an address cannot be specified by #pragma section in the CC-RL.
M0592144	Info	The *MACRO* cannot be converted. Because there is no matched macro.	The macro could not be converted because there is no corresponding macro in the CC-RL.
M0592145	Info	The label detected in the assembly code. Please correct label to appropriate content.	Only local labels can be written in an assembly-language function in the CC-RL. Modify the label to have suitable contents.
M0592146	Info	The language specification is dependent on 78K0.	The code needs to be reviewed when the device is changed from 78K0 to RL78. The code should be manually modified.

5. POINTS FOR CAUTION

If the C source program falls under any of the following items, it may not be possible for the CC-RL to correctly compile the converted C source program.

Table 5.1 Points for caution

No.	Item	CcnvCA78K0 Operation	CC-RL Operation in Response to Conversion Result	Reference Destination
1	When there is nested comment text	Conversion may not be performed successfully.	The range of the comment is invalid.	Macro Names
2	When a keyword cannot be detected because a ## operator is being used	No message is output and conversion is not performed.	Error E0520065 or another error will occur.	CONVERSION SPECIFICATIONS
3	When '?' is included in the section name for #pragma section	No message is output and conversion is not performed.	Error E0520014 will occur.	#pragma section
4	When a section name that does not exist in the CC-RL is specified for the section name of #pragma section	No message is output and conversion is not performed.	Warning W0523037 is output and the #pragma directive is ignored. There is a possibility that section allocation will fail and operation is not as expected.	#pragma section
5	When \n or \t is used in a string in __asm("string")	A control character is output without any change.	Error E0550249 will occur.	ASM Statements
6	When "/*" is included in an assembly-language comment (description after ";") within the range of #asm-#endasm	The assembly-language comment is output without any change.	A C-language comment ("/*") is given priority over an assembly-language comment (";") and the range of the comment is invalid.	ASM Statements
7	When a label is included in __asm( ) or the assembly-language code within #asm-#endasm	A message is output.	Error E0550213 will occur.	ASM Statements [Restrictions] of #pragma inline_asm in the CC-RL user's manual

No.	Item	CcnvCA78K0 Operation	CC-RL Operation in Response to Conversion Result	Reference Destination
8	When a 78K0-specific instruction or an instruction whose operation differs from that of RL78 is used in __asm() or the assembly-language code of #asm-#endasm	No message is output and instructions are output without change to the #pragma inline_asm function.	When a 78K0-specific instruction is used, an assembly error will occur. When an instruction whose operation differs from that of RL78 is used, operation may not be as expected.	ASM Statements
9	When there is #pragma interrupt and a description of the __interrupt keyword for the same function in a file	They are both converted into #pragma interrupt.	There will be duplicate #pragma directives and error E0523006 will occur.	Interrupt Handler
10	When parameters in function declarations __interrupt, __interrupt_brk, and __rtos_interrupt are omitted	A #pragma directive is output and function declarations are output without change.	Error E0523008 will occur since there is no void type specification.	Interrupt Handler Interrupt Handler for RTOS
11	When a macro or typedef is used in code for obtaining a type name, function name, variable name, etc.	A message is output and the program is output without being converted.	Error E0520020, E0520065, or another error will occur.	Interrupt Handler Interrupt Handler for RTOS Absolute Address Allocation Specification
12	When there is #pragma rtos_interrupt and a description of the __rtos_interrupt keyword for the same function in a file	They are both converted into #pragma rtos_interrupt.	There will be duplicate #pragma directives and error E0523006 will occur.	Interrupt Handler for RTOS
13	When the same address is specified for different variables in __directmap	No message is output.	Error E0541854 will occur.	Absolute Address Allocation Specification [Restrictions] of #pragma address in the CC-RL user's manual

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Apr 01, 2016	—	First Edition issued

CcnvCA78K0 C Source Converter User's Manual

Publication Date: Rev.1.00 Apr 01, 2016

Published by: Renesas Electronics Corporation

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

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

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

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

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

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

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

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

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

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

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

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

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

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0

CcnvCA78K0

Renesas Electronics Corporation

R20UT3684EJ0100