User's Manual

CcnvNC30

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 (CcnvNC30) 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 NC30, which is a C compiler for M16C Series and R8C Family, 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 NC30 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.
NC30	M3T-NC30WA V.6.00 C/C++ Compiler User's Manual (C/C++ Compiler Package for M16C Series and R8C Family)	REJ10J2188-0100
CC-RL	CC-RL Compiler User's Manual	R20UT3123EJ0102

TABLE OF CONTENTS

1. GENERAL

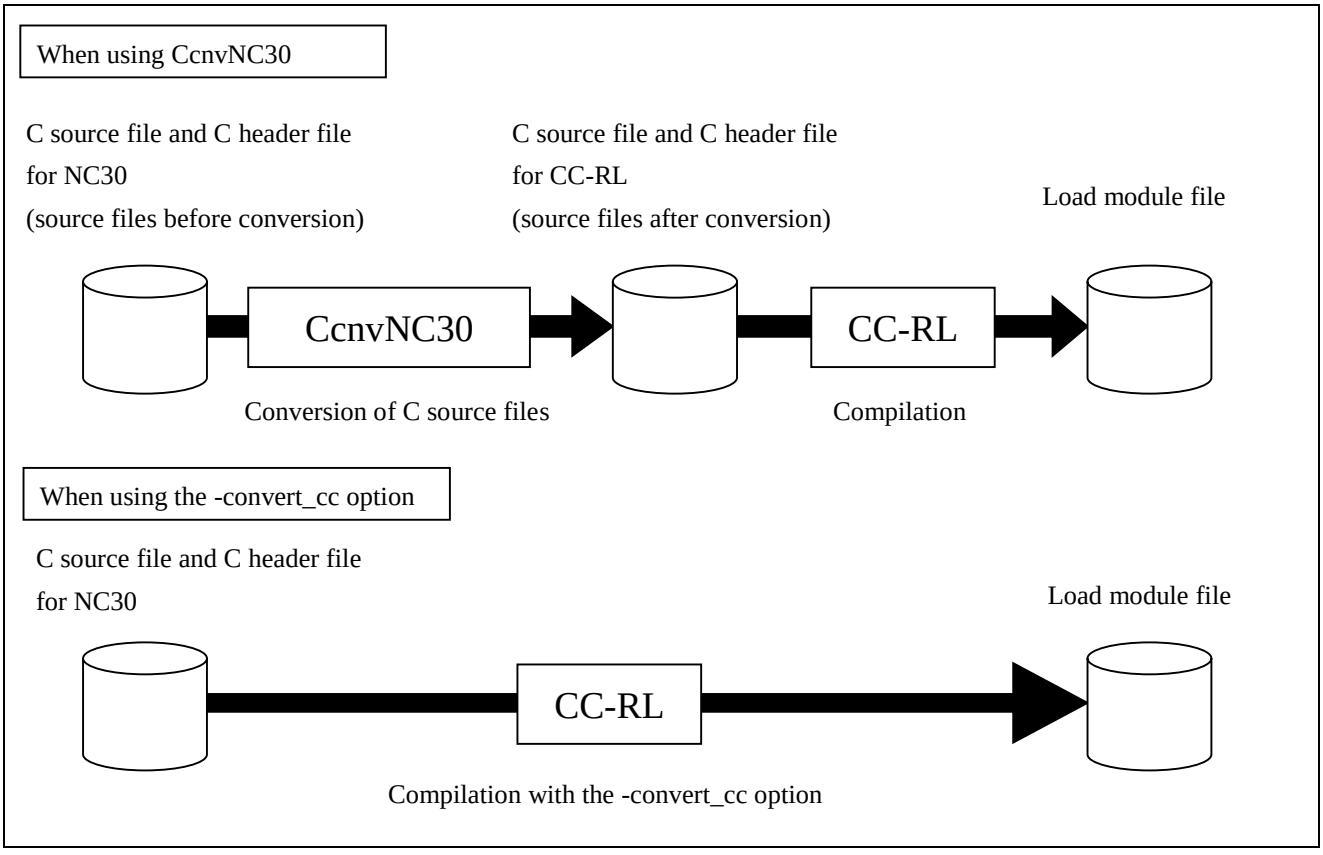
The CcnvNC30 is a C source converter that converts C source files created in a development environment using the NC30 which is a C compiler for the M16C series and R8C family microcontrollers into C source files that can be handled by the CC-RL which is a C compiler for the RL78 family microcontrollers. The extended functions for the NC30 written in C source files are converted so that they can be handled by the CC-RL.

The CC-RL has the -convert_cc option which internally converts extended functions of the NC30 in C source files into those of the CC-RL. The -convert_cc option of the CC-RL is useful when the files to be converted are the target of maintenance and so the future changes are to be made on a small scale or when evaluating how porting of code affects performance.

Use the CcnvNC30 in cases where C source code needs to be modified manually on a massive scale if the -convert_cc option of the CC-RL is used or where C source files for the CC-RL are required because there will be new features to be added.

CcnvNC30 supports the porting of C source files from the NC30 compiler to CC-RL.
Since we do not guarantee the correct operation of programs converted by CcnvNC30, be sure to check the operation of the C source files after conversion.

Figure 1.1 Comparison between CcnvNC30 and -convert_cc (CC-RL Option)



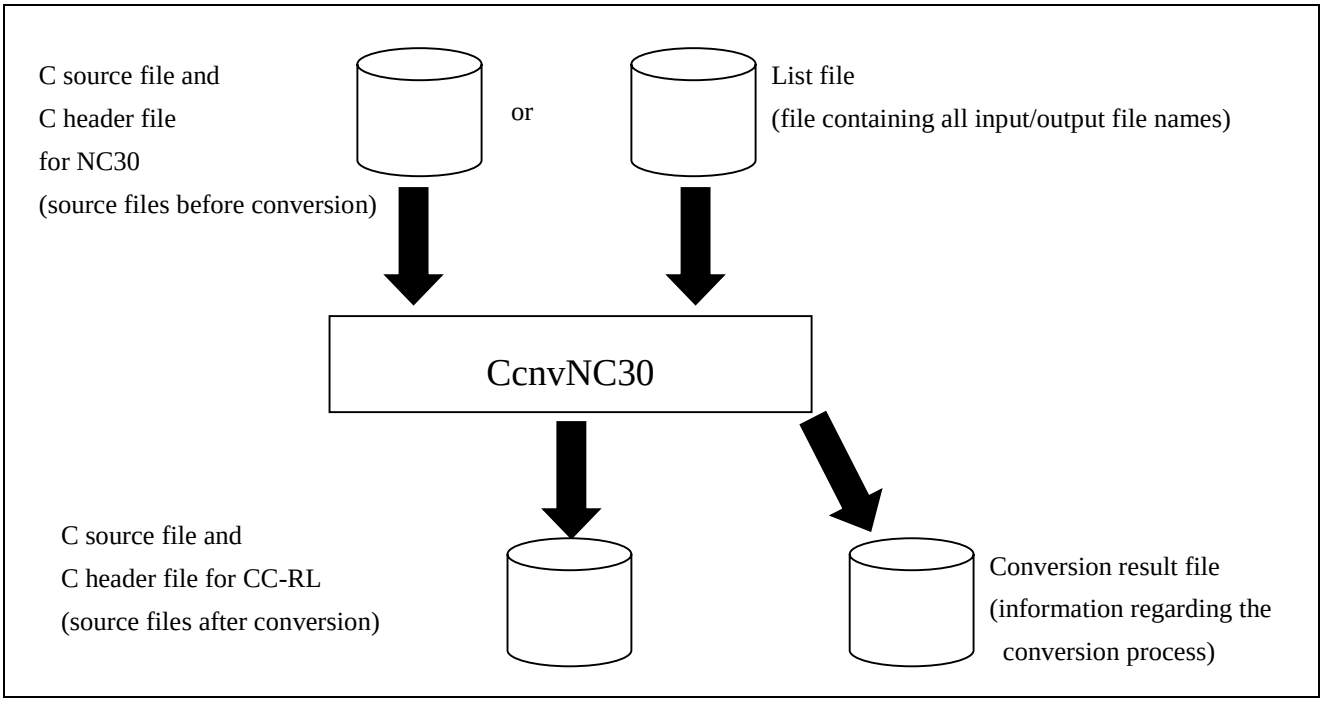
2. COMMAND REFERENCE

This section describes the processing flow in the CcnvNC30.

2.1 Overview

The CcnvNC30 converts extended language specifications (such as macro names, reserved words, #pragma directives, and extended functions) in C source programs for the NC30 into extended language specifications for the CC-RL. Then the CcnvNC30 generates C source files for the CC-RL.

Figure 2.1 Processing Flow in CcnvNC30



2.2 I/O Files

The I/O files of the CcnvNC30 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 NC30 is input and the converted C source file or C header file for the CC-RL is output. The version information of the CcnvNC30 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 ADDRESS p0 00E0H /* Port P0 register */

char c;

void main(void)

{

    c = p0;

}
```

(Output file: output.c)

```
/* NC30 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.
************************************************************************/

//[CcnvNC30] #pragma ADDRESS p0 00E0H /* Port P0 register */

#pragma address p0=0x00E0

char c;

void main(void)

{

   c = p0;

}
```

2.3 Conversion Result

The CcnvNC30 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 CcnvNC30

```
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 ADDRESS p0 00E0H /* Port P0 register */
char c;
void main(void)
{
    c = p0;
}
```

(Standard output)

```
NC30 C Source Converter Vx.xx.xx.xx [dd Mmm yyyy]


input.c(1):M0593113:[Change]#pragma address has been changed to syntax of CC-RL.
input.c(1):M0593146:[Info]The language specification dependent on R8C or M16C.


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

2.4 Method for Manipulating

Input on the command line should be made as follows.

CcnvNC30[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 CcnvNC30 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 CcnvNC30.
-h	This option displays the descriptions of CcnvNC30 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.
-R8C	This option performs conversion with C source files regarded as those for the R8C.

-V

This option displays the version information of CcnvNC30.

[Specification format]

-V

・Interpretation when omitted
The version information of CcnvNC30 is not displayed.

[Detailed description]

- This option outputs the version information of CcnvNC30 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]

```
>CcnvNC30    -V
```

-h

This option displays the descriptions of CcnvNC30 options.

[Specification format]

```
-h
```

・Interpretation when omitted
The descriptions of CcnvNC30 options are not displayed.

[Detailed description]

- This option outputs the descriptions of CcnvNC30 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]

```
>CcnvNC30    -h
```

-C

This option specifies the Japanese character code.

[Specification format]

```
-c={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.

sjis	SJIS
euc_jp	EUC (Japanese)

[Example of use]

```
>CcnvNC30  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={sjis | euc_jp}] [-A] input-file-name output-file-name
[-c={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]

```
>CcnvNC30    -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]

```
>CcnvNC30    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]

```
>CcnvNC30    input.c    -o=output.c    -r=input.txt
```

-A

This option performs conversion with the -fansi option (which is an ANSI-compliant option of NC30) enabled.

[Specification format]

```
-A
```

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

[Detailed description]

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

```
far,  near,  inline,  asm
```

・ Specify this option if the -fansi option is used in the NC30 development environment before conversion.

[Example of use]

```
>CcnvNC30  input.c  -o=output.c  -A
```

-R8C

This option performs conversion with C source files regarded as those for the R8C.

[Specification format]

```
-A
```

・Interpretation when omitted
Conversion is performed with C source files regarded as those for the M16C.

[Detailed description]

- When this option is specified, the _far or far keyword is handled as the _near or near keyword.
- When this option is specified, #pragma SPECIAL is disabled, a message is output and #pragma SPECIAL is deleted.
- Specify this option in accordance with the NC30 development environment before conversion.

[Example of use]

```
>CcnvNC30  input.c  -o=output.c  -R8C
```

3. CONVERSION SPECIFICATIONS

This section shows the conversion specifications of the CcnvNC30.

- Correct operation is not guaranteed when a C source program that is syntactically incorrect for the NC30 is input.
- The contents included in comments, strings, and character constants 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.
- Code that is dependent on the device should be manually modified after conversion.
- 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
- Default Parameters
- Concatenation of Wide String and Character Constant
- #pragma SPECIAL
- #pragma SECTION
- ASM Statements
- Interrupt Handler
- Absolute Address Allocation Specification
- Intrinsic Functions
- Other #pragma Directives
- Standard Library Functions

3.1 Macro Names

The macros supported in the NC30 are converted as follows. If there is no corresponding macro in the CC-RL, the CcnvNC30 outputs a message. Standard library macros that are supported only by the NC30 are not converted and no message is output. They are handled as user-defined macros in the CC-RL.

Table 3.1 Conversion of Macro Names

NC30 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.
NC30	Not converted	A message is output. Handled as a user-defined macro in the CC-RL.
M16C	Not converted	A message is output. Handled as a user-defined macro in the CC-RL.
__R8C__	Not converted	A message is output. Handled as a user-defined macro in the CC-RL.
__cplusplus	Not converted	A message is output. Handled as a user-defined macro in the CC-RL.
MB_LEN_MAX	Not converted	<limits.h> A message is not output. Handled as a user-defined macro in the CC-RL.
LC_ALL LC_COLLATE LC_CTYPE LC_MONETARY LC_NUMERIC LC_TIME	Not converted	<locale.h> A message is not output. Handled as a user-defined macro in the CC-RL.
SIGABRT SIGFPE SIGILL SIGSEGV SIG_DFL SIG_ERR SIG_IGN	Not converted	<signal.h> A message is not output. Handled as a user-defined macro in the CC-RL.

NC30 Macro Name	After Conversion	Remarks
_IOFBF _IOLBF _IONBF BUFSIZ FILENAME_MAX FOPEN_MAX SEEK_CUR SEEK_END SEEK_SET TMP_MAX stderr stdin stdout	Not converted	<stdio.h> A message is not output. Handled as a user-defined macro in the CC-RL.
MB_CUR_MAX	Not converted	<stdlib.h> A message is not output. Handled as a user-defined macro in the CC-RL.
CLOCKS_PER_SEC	Not converted	<time.h> 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

NC30 Reserved Word	After Conversion	Remarks
wchar_t	Not converted	"typedef unsigned short wchar_t" is output at the beginning of a file. A message is output for wchar_t that was written first.
_inline	__inline	
inline	__inline	Converted only when the -A option is invalid.
restrict	Deleted	
_ext4mptr	Deleted	
_near	__near	
near	__near	Converted only when the -A option is invalid.
_far	__far (without -R8C) __near (with -R8C)	
far	__far (without -R8C) __near (with -R8C)	Converted only when the -A option is invalid.

3.3 Default Parameters

Though default values can be defined for parameters of functions in the NC30 (similar to as the C++ facility), default parameters cannot be specified in the CC-RL. The CcnvNC30 does not convert default parameters and outputs a message. However, default parameters cannot be recognized correctly if no storage-class specifier, type specifier, or type qualifier could be detected when a typedef name is used as the type name at the function declaration.

[Examples]

Pattern	Example	Remarks
Pattern 1	int func1(int a, char b=1);	A message is output.
Pattern 2 (variable name is omitted)	int func2(int, int=2);	A message is output.
Pattern 3 (type specifier is omitted)	int func3(unsigned a=3);	A message is output.
Pattern 4 (structure)	struct S {int i;} s; int func4(struct S param=s);	A message is output.
Pattern 5 (typedef is used)	typedef int INT16; int func5(INT16 param=2);	Cannot be recognized as a default parameter. No message is output.

3.4 Concatenation of Wide String and Character Constant

When a character string literal (e.g. "abc") and a wide string literal (e.g. L"def") are next to each other, the constants are handled differently between the NC30 and CC-RL. Also, a character constant of two or more characters (e.g. 'ab') or an integer character constant for a wide character is handled differently between the NC30 and CC-RL. In these cases, the CcnvNC30 outputs a message but does not perform conversion so the code has to be manually modified at the locations where a message was output.

- When a character string literal is next to a wide string literal

  <NC30>

  They are concatenated without their types being changed to match each other.

L"abc""def"	00H	61H	00H	62H	00H	63H	64H	65H	66H	00H				
"abc"L"def"	61H	62H	63H	00H	64H	00H	65H	00H	66H	00H	00H			

  <CC-RL>

  When a character string literal and a wide string literal are concatenated, it is handled as a wide string.

L"abc""def"	00H	61H	00H	62H	00H	63H	00H	64H	00H	65H	00H	66H	00H	00H
"abc"L"def"	00H	61H	00H	62H	00H	63H	00H	64H	00H	65H	00H	66H	00H	00H

- Character constant of two or more characters

  The character constant does not have the same value.

	NC30	CC-RL
'ab'	0x61	0x6162
L'ab'	0x6162	0x0061

3.5 #pragma SPECIAL

The functions for calling special page subroutines are replaced with the callt function.

The format of the NC30 is as follows.

```
#pragma SPECIAL number function-name
or
#pragma SPECIAL function-name(vect=number)
```

The format of the CC-RL is as follows.

```
#pragma callt [ ( ] function-name [ , … ] [ ) ]
```

- When the -R8C option is specified, #pragma SPECIAL is deleted.
- Up to 32 callt functions can be specified. If there are more than 32 specifications of #pragma SPECIAL, a compile error will occur after conversion.

[Examples]

Pattern 1 (without -R8C)	Before conversion	#pragma SPECIAL 20 func
	After conversion	#pragma callt func
Pattern 2 (without -R8C)	Before conversion	#pragma SPECIAL func(vect=20)
	After conversion	#pragma callt func
Pattern 3 (with -R8C)	Before conversion	#pragma SPECIAL func(vect=20)
	After conversion	//[CcnvNC30] #pragma SPECIAL func(vect=20)

3.6 #pragma SECTION

#pragma SECTION requires the section name to be converted because the section names differ between the NC30 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 CcnvNC30 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 NC30 is as follows.

```
#pragma SECTION section-name changed-section-name
```

The format of the CC-RL is as follows.

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

- 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 CcnvNC30 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 rom MY_DATA
	After conversion	#pragma section const MY_DATA
Pattern 2 (Replacement is not possible)	Before conversion	#pragma SECTION interrupt MY_CODE
	After conversion	#pragma section interrupt MY_CODE
	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

NC30 Section Name	Description	CC-RL Section Type	CcnvNC30 Operation Corrective Action after Conversion
program	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".
data	Segment for data area (initialized)	data	Conversion is not performed.
			The section name in the CC-RL is "changed section name + _n" or "changed section name + _f". Specify the section for mapping ROM to RAM with the link option -ROm.
bss	Segment for data area (uninitialized)	bss	Conversion is not performed.
			No action is required. The section name in the CC-RL is "changed section name + _n" or "changed section name + _f".
rom	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" or "changed section name + _f".
interrupt	Segment for compatibility	None	Conversion is not performed.
			Delete #pragma. The section name cannot be changed in the CC-RL.

3.7 ASM Statements

The _asm( ) or asm( ) function or #pragma asm-#pragma endasm is used to write assembly-language code within functions in the NC30, whereas inline expansion is performed for the assembly-language functions declared in #pragma inline_asm in the CC-RL. The CcnvNC30 creates the inline_asm function that executes assembly instructions in the _asm( ) or asm( ) function or the range between #pragma asm and #pragma 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 NC30 is as follows.

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

or

```
_asm(/* comment */);
_asm("assembly-language code");
_asm("assembly-language code", parameter1);
_asm("assembly-language code", parameter1, parameter2);
```

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 RL78 and R8C or M16C, 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.
- The _asm( ) function is always converted. The asm( ) function, on the other hand, is converted only when the -A option is not specified.
- If a label is in the range between #pragma asm and #pragma endasm or in the _asm(asm) function, the CcnvNC30 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 #pragma asm–#pragma endasm or the _asm(asm) function, the CcnvNC30 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 CcnvNC30 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( ) or 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 #pragma asm and #pragma 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)`
		`{`
		`  #pragma asm`
		`  nop`
		`  #pragma 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();
		}

3.8 Interrupt Handler

#pragma interrupt of the NC30 is converted into #pragma interrupt of the CC-RL.

The format of an interrupt function of the NC30 is as follows.

```
#pragma interrupt [/B | /E | /V] [interrupt-vector-number] function-name
 or
#pragma interrupt [/B | /E] function-name [(vect=interrupt-vector-number)]
```

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
```

- "/B" is converted into "bank=RB1" and "/E" is converted into "enable=true". When "/B" and "/E" are specified simultaneously, #pragma interrupt is deleted.
- When "/V" is specified, #pragma interrupt is deleted.
- When an interrupt vector number is specified, it is converted into "(vect=address)" and #include "iodefine.h" is output separately. The value of an interrupt vector number of the NC30 directly becomes an address of the CC-RL. However, a message is output because operation cannot be performed correctly because the device is changed. The value should be manually modified.
- Though #pragma interrupt can be written more than once for the same function in the NC30, this causes a compile error to occur in the CC-RL. Duplicate #pragma directives should be manually deleted.

[Examples]

Pattern 1	Before conversion	#pragma interrupt /V func
	After conversion	//[CcnvNC30] #pragma interrupt /V func
Pattern 2	Before conversion	#pragma interrupt /B /E 10 func
	After conversion	//[CcnvNC30] #pragma interrupt /B /E func(vect=10)
Pattern 3	Before conversion	#pragma interrupt /E 10 func void func(void) { }
	After conversion	#pragma interrupt func(vect=10, enable=true) void func(void) { }
Pattern 4	Before conversion	#pragma interrupt /B func(vect=10) void func(void) { }
	After conversion	#pragma interrupt func(vect=10, bank=RB1) void func(void) { }
Pattern 5	Before conversion	#pragma interrupt func
	After conversion	#pragma interrupt func

3.9 Absolute Address Allocation Specification

#pragma address of the NC30 is converted into #pragma address of the CC-RL.

The format of the NC30 is as follows.

#pragma address variable-name location-address

The location address can be written in binary, octal, decimal, or hexadecimal notation.

'B' or 'b', 'O' or 'o', and 'H' or 'h' are appended at the end of the number for a binary number, an octal number, and a hexadecimal number, respectively.

An expression such as "variable address + offset" can be written for the location address.

The format of the CC-RL is as follows.

#pragma address variable-name = location-address

The location address can be written as a binary, octal, decimal, or hexadecimal number in C-language notation. An expression cannot be written.

- The location address is converted into C-language notation with the same radix as before conversion.
- When an expression is written as the location address, a message is output and conversion is not performed.
- Since the memory map is different between the RL78 and R8C or M16C, a message is output. The code should be manually modified in accordance with the memory map of the RL78.
- Access to an SFR needs to be reviewed because the device is different. The code should be manually modified.

[Examples]

Pattern 1	Before conversion	#pragma address i 01010101b
	After conversion	#pragma address i= 0b01010101
Pattern 2	Before conversion	#pragma address i 002000O
	After conversion	#pragma address i=02000
Pattern 3	Before conversion	#pragma address i 500
	After conversion	#pragma address i=500
Pattern 4	Before conversion	#pragma address i 400h
	After conversion	#pragma address i=0x400
Pattern 5	Before conversion	#pragma address i 0400H + _OFFSET
	After conversion	#pragma address i 0400H + _OFFSET

3.10 Intrinsic Functions

Since intrinsic functions of the NC30 do not match the functionality of those of the CC-RL, a message is output and they are not converted into intrinsic functions of the CC-RL. They are handled as user-defined functions in the CC-RL.

Table 3.4 Conversion of Intrinsic Functions

NC30 Intrinsic Function	Remarks
abs_b, abs_w, dadc_b, dadc_w, dadd_b, dadd_w, div_b, div_w, divu_b, divu_w, divx_b, divx_w, mod_b, mod_w, modu_b, modu_w, not_b, not_w, neg_b, neg_w, dsbb_b, dsbb_w, movll, movlh, movhl, movhh, rmpa_b, rmpa_w, smovf_b, smovf_w, sha_b, sha_w, sha_l, shl_b, shl_w, shl_l, smovb_b, smovb_w, sstr_b, sstr_w, rolc_b, rolc_w, rorc_b, rorc_w, rot_b, rot_w	A message is output and conversion is not performed.

3.11 Other #pragma Directives

Conversion specifications for other #pragma directives are shown here.

Table 3.5 Conversion of Other #pragma Directives

NC30 #pragma Directive	After conversion	Remarks
#pragma rom	Deleted	Not supported in the CC-RL.
#pragma struct	Deleted	Not supported in the CC-RL.
#pragma ext4mptr	Deleted	Not supported in the CC-RL.
#pragma bitaddress	Deleted	Not supported in the CC-RL.
#pragma intcall	Deleted	Not supported in the CC-RL.
#pragma parameter	Deleted	Not supported in the CC-RL.
#pragma __asmmacro	Deleted	Not supported in the CC-RL.
#pragma jsra	Deleted	Not supported in the CC-RL.
#pragma jsrw	Deleted	Not supported in the CC-RL.
#pragma page	Deleted	Not supported in the CC-RL.
#pragma stacksize	Deleted	Not supported in the CC-RL.
#pragma istacksize	Deleted	Not supported in the CC-RL.
#pragma creg	Deleted	Not supported in the CC-RL.
#pragma sectaddress	Deleted	Not supported in the CC-RL.
#pragma entry	Deleted	Not supported in the CC-RL.
#pragma almhandler	Deleted	Not supported in the CC-RL.
#pragma inthandler	Deleted	Not supported in the CC-RL.
#pragma handler	Deleted	Not supported in the CC-RL.
#pragma cychandler	Deleted	Not supported in the CC-RL.
#pragma task	Deleted	Not supported in the CC-RL.
#pragma bit	Deleted	Not supported in the CC-RL.
#pragma sbda	Deleted	Not supported in the CC-RL.

3.12 Standard Library Functions

The NC30 has standard library functions that are not supported by the CC-RL. For any of these functions, the CcnvNC30 does not output a message and does not perform conversion. They are handled as user-defined functions by the CC-RL.

- Do not use the CcnvNC30 to convert the header file of the standard libraries for the NC30 and make the CC-RL handle the converted header file. Use the header file of the standard libraries for the CC-RL.
- 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 NC30	After Conversion	Remarks
<locale.h> setlocale, localeconv	Not converted	Not supported in the CC-RL. Handled as a user function in the CC-RL.
<signal.h> signal, raise	Not converted	Not supported in the CC-RL. Handled as a user function in the CC-RL.
<stdio.h> fflush, fprintf, fscanf, vfprintf, vsprintf, fgetc, fgets, fputc, fputs, getc, putc, ungetc, fread, fwrite, clearerr, feof, ferror	Not converted	Not supported in the CC-RL. Handled as a user function in the CC-RL.
<stdlib.h> exit, mblen, mbtowc, wctomb, mbstowcs, wcstombs	Not converted	Not supported in the CC-RL. Handled as a user function in the CC-RL.
<string.h> strcoll, strxfrm	Not converted	Not supported in the CC-RL. Handled as a user function in the CC-RL.
bcopy, bzero, memicmp, stricmp, strnicmp	Not converted	Handled as a user function in the CC-RL.

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

The difference of extended functions whose operations vary when the CC-RL's option -convert_cc is used and when conversion is performed by the CcnvNC30 is shown here.

Table 3.7 Different Operation from -convert_cc=nc30 Option of CC-RL

NC30 Extended Function	Operation when -convert_cc=nc30 Option is Used	Conversion by CcnvNC30
#pragma interrupt	When a description is in a different format from that of the CC-RL, the #pragma directive is deleted and a warning message is output. The interrupt vector number needs to be manually modified to that for the RL78.	Converted to the format of the CC-RL. Since specifications of the interrupt vector differ between devices, a message is output. The interrupt vector number needs to be manually modified to that for the RL78.
#pragma asm : #pragma endasm	The #pragma directives are deleted and a warning message is output.	#pragma inline_asm and a function definition are output, and #pragma asm–#pragma endasm is converted into a newly generated function call. Since the instruction set is different between the RL78 and R8C or M16C, a message is output. The assembly-language code needs to be manually modified to that for the RL78.
_asm( ), asm( )	Recognized as a normal function call. It needs to be manually modified to the inline_asm function. The assembly-language code needs to be manually modified to that for the RL78.	#pragma inline_asm and a function definition are output for each _asm( ) and asm( ). Calls for _asm( ) and asm( ) are converted into newly generated function calls. Since the instruction set is different between the RL78 and R8C or M16C, a message is output. The assembly-language code needs to be manually modified to that for the RL78.
NC30 M16C __R8C__	The macro is enabled (a space is defined).	Conversion is not performed and a message is output.

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, 0593, 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 NC30 and CC-RL specifications. Each case should be confirmed individually.

4.4 Messages

The messages output by the CcnvNC30 are as follows.

4.4.1 Internal Errors

Table 4.3 Internal Errors

Number	Message	Description
C0593*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
E0593001	Multiple input files are not allowed.	Only one input file can be specified. Use the list file to specify multiple input files.
E0593002	The *option* option cannot have an argument.	An argument was specified for an option that should not have arguments.
E0593003	The *option* option requires an argument.	No argument was specified in an option that requires arguments.
E0593004	The *option* option is specified more than once.	Only one option can be specified at one time.
E0593005	Requires an output file.	The output file corresponding to the input file was not specified.
E0593006	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.
E0593007	Failed to write a result of conversion file *file*.	The folder name may be incorrect.
E0593008	Failed to write an output file *file*.	The folder name may be incorrect.
E0593009	Failed to read a list file *file*.	The folder name may be incorrect.
E0593010	Syntax errors in list file *file*.	The description of the list file is not correct.
E0593011	File name is corrupted.	There are duplicate file names among the input file, output file, and conversion result output file.
E0593012	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.
E0593013	Invalid argument for the *option* option.	The argument specification is invalid or the specified file name has exceeded 260 characters.
E0593101	Illegal syntax in *string*.	Conversion could not be performed because there was a syntax that is not allowed in the NC30. Modify the input file.
E0593102	Can not add inline function for assembly.	The number of inline functions for assembly has exceeded the upper limit. Modify the input file.
E0593103	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
W0593051	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.
W0593052	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.
W0593053	Invalid *option* option.	An invalid option was specified. Ignore the option.
W0593054	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.
W0593055	Requires an input file.	The list file specified by the "-l" option is missing an input file specification.
W0593151	*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
M0593111	Change	*String1* was converted into *string2*.	The token was converted.
M0593113	Change	'*String*' has been changed to syntax of CC-RL.	Since the description format differs between the NC30 and CC-RL, the description format is changed to that of the CC-RL.
M0593123	Insert	Inserted *string*.	A description in accordance with the CC-RL format was added.
M0593124	Insert	Add inline function *string* for assembly.	An inline function for assembly was generated.
M0593131	Delete	*String* was deleted.	The description format is not available in the CC-RL. The description was deleted.

Number	Information Type	Message	Description
M0593142	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.
M0593144	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.
M0593145	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.
M0593146	Info	The language specification is dependent on R8C or M16C.	The code needs to be reviewed when the device is changed from R8C or M16C 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	CcnvNC30 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 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
4	When \n or \t is used in a string in _asm("string") or asm("string")	A control character is output without any change.	Error E0550249 will occur.	ASM Statements
5	When "/*" is included in an assembly-language comment (description after ";") within the range of #pragma asm – #pragma 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
6	When a label is included in _asm( ), asm( ), or the assembly-language code within #pragma asm – #pragma endasm	A message is output.	Error E0550213 will occur.	ASM Statements [Restrictions] of #pragma inline_asm in the CC-RL user's manual
7	When an ASM statement is written	A message is output. The statement is output without change to the #pragma inline_asm function.	An assembly error will occur because the instruction set is different.	ASM Statements

No.	Item	CcnvNC30 Operation	CC-RL Operation in Response to Conversion Result	Reference Destination
8	When an interrupt vector number is specified in an interrupt function	A message is output and the interrupt vector number is converted into "(vect=address)".	Since the specifications of interrupts are different between the RL78 and R8C or M16C, operation may not be as expected.	Interrupt Handler
9	When #pragma address is used	A message is output and it is converted into the format for the CC-RL.	Since the memory map is different between the RL78 and R8C or M16C, operation may not be as expected.	Absolute Address Allocation Specification

Revision Record

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

CcnvNC30 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

CcnvNC30

Renesas Electronics Corporation

R20UT3685EJ0100