To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

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

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

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

===== Be sure to read this note. =====

C Compiler Package for R32C/100 Series
V.1.01 Release 00
Release note
(Rev.2.0)

Renesas Solutions Corporation

Feb 1, 2009

Abstract

Welcome to C Compiler Package for R32C/100 Series V.1.01 Release 00. This document contains supplementary descriptions to User's Manual. When you read certain items in the User's manual, please read this document as well.
Also, this document contains a License Agreement in the last. Please read it before using. By using the software, you are accepting and agreeing to such terms.

1. Precautions on Product

When using the compiler, please be sure to follow the precautions and suggestions described below.

1.1. Precaution of MCU-Dependent Code

1.1.1. R32C Precautions regarding the R32C interrupt control register

When the "-O5" optimizing options is used, the compiler generates in some cases "BTSTC" or "BTSTS" bit manipulation instructions. In R32C/100, the "BTSTC" and "BTSTS" bit manipulation instructions are prohibited from rewriting the contents of the interrupt control registers.
However, the compiler does not recognize the type of any register, so, should "BTSTC" or "BTSTS" instructions be generated for interrupt control registers, the assembled program will be different from the one you intend to develop.
When the "-O5" optimizing options is used in the program shown below, a "BTSTC" instruction is generated at compilation, which prevents an interrupt request bit from being processed correctly, resulting in the assembled program performing improper operations.

- Example of occurrence

```
#pragma ADDRESS ta0ic_addr 006CH /* Timer A0 interrupt control register */

struct {
char ilvl :3;
char ir :1; /* An interrupt request bit */
char dmy :4;
} ta0ic;

void wait_until_IR_is_ON(void)
{
        while (ta0ic.ir == 0)       /* Waits for ta0ic.ir to become 1 */
        {
        ;
        }
        ta0ic.ir = 0;               /* Returns 0 to ta0ic.ir when it becomes 1 */
}
```

- Example of occurrence

(1) Optimization options other than "-O5" are used". When you use the optimization option of "-O5", please use together with"-O5A."

(2) Add an asm function to disable optimization locally, as shown in the example below.

```
void wait_until_IR_is_ON(void)
{
        while (ta0ic.ir == 0)       /* Waits for ta0ic.ir to become 1 */
        {
                asm();
        }
        ta0ic.ir = 0;               /* Returns 0 to ta0ic.ir when it becomes 1 */
}
```

1.1.2. Precautions about access of SFR area

You may need to use specific instructions when writing to or reading registers in the SFR area. Because the specific instruction is different for each model, see the User's Manual for the specific Machine. These instructions should be used in your program using the asm function.

1.2. Precautions about C Compiler

1.2.1. Precautions about assign of array

- Description

  When a constant as a subscript is assigned to the same element of array more than once, incorrect code may be generated for the assignment.

- Conditions

  This problem occurs if the following conditions are all satisfied:

  (1) Any one compile option is selected out of -O4, -O5, -OR_MAX(-ORM), and -OS_MAX(-OSM).
  (2) The subscript of the element of array assigned is a constant.
  (3) Two or more assignment expressions of (2) exist in one function.
  (4) The assignment expression after the second of (3) refers to an element of array using a variable with an initial value as a subscript, and the element of array is the same as the assignment-destionation.
  (5) (2), (3), and (4) show all the same elements of array.
  (6) The function call or the pointer reference does not exist between (3) and (4).

  Example of occurrence)

```
int   ary[3];
int   index=2;

void main(void) {
   ary[2] = 1;                    /* Conditions   (2),(3), and (5) */
    ;
    ;
   ary[2] = ary[index] + 1 ;     /* Conditions   (2),(3),(4), and (5) */
}
```

  In the above example, the cord for " ary[2] = 1;" is not generated.

- Workaround

  This problem can be avoided in either of the following ways:

  (1) Never select compile option -O4, -O5, -OR_MAX(-ORM), and -OS_MAX(-OSM).
  (2) Place a dummy asm function between generation conditions (4).

  Example)

```
int   ary[3];
int   index=2;

void main(void) {
   ary[2] = 1;
    ;
    ;
   asm( );                        /* asm function placed */
   ary[2] = ary[index] + 1;
}
```

1.2.2. Precautions about the search of an include file

If you give a file to include together with a drive name in the #include line, and attempt to compile the file from a directory different from the one in which the file to compile is present, instances may occur in which the file to include cannot be searched.

1.2.3. Precautions to be taken when using #pragma ASM/ENDASM and asm()

(1) Regarding debug information when using #pragma ASM outside functions, if you write #pragma ASM anywhere outside functions, no C source line information will be output. For this reason, information regarding descriptions in #pragma ASM to #pragma ENDASM, such as error message lines when assembling or linking and line information when debugging, may not be output normally.

(2) C compilers generate code of arguments to be passed via registers and of register variables by analyzing their scopes. However, if manipulations of register values are described using inline assemble functions (such as #pragma ASM / #pragma ENDASM directives and asm function), C compilers cannot hold information on the scopes of the above-mentioned arguments and register variables. So, be sure to save and recover register contents on and from the stack when registers are loaded using inline assemble functions described above.

1.2.4. Precautions about regarding the preprocessing directive #define

To define a macro which will be made the same value as the macro ULONG_MAX, always be sure to add the prefix UL.

1.2.5. Precaution about the macro definition

- Description

  If the name of a macro itself is used in the content of a macro definition and the defined macro is specified in an argument to other function-like macro, macro replacement cannot be executed correctly.

- Example of occurrence

```
int a = 10;
#define a a + a              // macro name 'a'
#define p(x,y) x + y

void func( void )
{
    int i = p( a, a ) ;      //  results in i = 80
}                            //  (i = 40 is correct)
```

- Workaround

  Make sure the macros passed to the arguments to function-like macros are defined with a name that is not used in the macro definition.

```
int a = 10;
#define b a + a        //  Change to a macro name that is not 'a'
#define p(x,y) x + y

void func( void )
{
    int i = p( b, b ) ;
}
```

1.2.6. Precautions about conditional compilation directive #if.

- Description

  If a constant expression of #if directive is a shift whose left operand is a negative value and right operand is a value of unsigned type, the result of the shift cannot be determined to be good or not correctly.

- Example of occurrence

```
void func( void )
{
      char a;

#if (-1 << 1U) > 0        //   Determined to be true
      a=1;                //   (-1 << 1U) is -2, so that it correctly is false
#else
      a=2;
#endif
}
```

- Workaround

  If the left operand of a shift is a negative value, change the right operand of that shift to a value of signed type.

```
int main( void )
{
      char a;

#if (-1 << 1) > 0     //   Disuse of the suffix U changes
      a=1;            //   the right operand of a shift to signed type.
#else
      a=2;
#endif
}
```

1.2.7. Precautions about malloc()

Memory management function malloc of the NC100 cannot secure the area of 64KB or more at a time.
(When compilation option fint_16(-fI16) is selected.)

1.3. Precautions about MS-Windows

1.3.1. Precautions about environment of operation

(1) C Compiler Package operates under Windows 98, Windows NT 4.0 or later. It does not work under Windows 95 and Windows NT 3.5x or earlier.

(2) If in Windows NT environment the command prompt size is set to other than "80 x 25", the command prompt size will change frequently as you start the compiler. Make sure the command prompt size is set to "80 x 25".

1.3.2. Suggestions Concerning File Names

The file names that can be specified are subject to the following restrictions:

(1) Directory and file names that contain kanji cannot be used.

(2) Only one period (.) can be used in a file name.

(3) Network path names cannot be used. Assign the path to a drive name.

(4) Keyboard shortcuts cannot be used.

(5) The "..." symbol cannot be used as a means of specifying two or more directories.

1.3.3. Precautions about virus check programs

If the virus check program is memory-resident in your computer, C Compiler Package may not start up normally. In such a case, remove the virus check program from memory before you start C Compiler Package.

2. Installing C Compiler Package

2.1. Before installing C Compiler Package

(1) Please carefully read the "License Agreement" and “Release Note” included with your product before using C Compiler Package. If you've installed this product in your computer, it is assumed that you've agreed to the provisions stipulated in the License Agreement.

(2) You need to input a license ID in the middle of installation. Before you start installing C Compiler Package, check your license ID.

2.2. About entering user information

The data you input in the intermediate of installation is necessary to create a file for user registration. For information on our policy concerning the protection of personal information, please refer to the Renesas Technology Homepage.

2.3. About entering user registration

To be eligible for upgrade information, technical support, and other services, you must be registered as a user with Renesas Technology Corporation. Unless you are a registered user, the said services cannot be received.

2.4. Required System Configuration

Host computer	IBM[1] PC compatible machine
OS	Windows[2] 98, Windows Me, Windows NT 4.0, Windows 2000 or Windows XP
Memory capacity	512 MB or more recommended.
Hard disk drive	200 MB or more
Display	SVGA or higher
I/O device	CD-ROM drive
Others	Pointing device (e.g. a mouse)

[1] IBM is a registered trademark of International Business Machines Corporation.
[2] Windows, Windows NT are registered trademarks of Microsoft Corporation.

2.5. Installation Procedure

Insert the CD-ROM of the C Compiler Package into the CD drive of your computer, and High-performance Embedded Workshop Install Manager will start up automatically. Follow the messages displayed by High-performance Embedded Workshop Install Manager as you install the C Compiler Package.
If High-performance Embedded Workshop Install Manager does not start up automatically, execute "HewInstMan.exe" included in the CD-ROM.

```
D:\> HewInstMan.exe

        The drive name "D:" differs with each PC used.
```

Note, however, that before High-performance Embedded Workshop Install Manager starts up, all other applications must be closed.

2.6. Restrictions on installation of High-performance Embedded Workshop

(1) Directory and file names that contain multi-byte character (ex. Japanese-Kanji) space character cannot be used.
(2) If [Renesas] does not appear in the [Program] folder of the [Start] menu of Windows, restart the operating system.
(3) If your installation terminates abnormally during the installation, restart the host computer and install HEW V.4.02 again.
(4) Make sure that High-performance Embedded Workshop V.4.02 is installed in a different directory than the one where the earlier version V.1.x is installed.
(5) For the notice regarding HEW, see the following
http://tool-support.renesas.com/eng/toolnews/hew/hew.html

2.7. Uninstalling programs

The installed programs can be uninstalled according to the following instructions. Begin the un-installation after closing all the applications.
(1) Select [Control Panel] on the Windows [Start] menu.
(2) Select the [Add or Remove Programs] icon.
(3) Click [High-performance Embedded Workshop] on the [Change or Remove Programs] tab and click the [Remove] button.
(4) Follow the instruction displayed on the screen.

2.8. About the AutoUpdater

The AutoUpdater will start and station into PC automatically.
The AutoUpdater is an utility that watch the Renesas HomePage periodically and detects the renewal of the installed development tools.

2.9. Constitution of start menu

After installation, the folders and shortcuts that showed them below will be registered to the [start] -> [Programs] -> [Renesas].

High-performance Embedded Workshop
R32C-100 Series C Compiler V.1.01 Release 00
Renesas Tools HomePage
Renesas AutoUpdate
Renesas AutoUpdate Mnual

2.10. Startup or termination of program

2.10.1. Startup and termination of the High-performance Embedded Workshop

- Startup

  Click [High-performance Embedded Workshop] in the [High-performance Embedded Workshop] folder in the [Renesas] folder in the [Program] folder of the Windows [Start] menu.

- Termination

  Click [Exit] on the [File] menu.

2.10.2. Launch Manual Navigator

- Startup

  Click [Manual Navigator] in the [High-performance Embedded Workshop] folder in the [Renesas] folder in the [Program] folder of the Windows [Start] menu.

- Termination

  Termination: Click [Exit] on the [File] menu.

- Note

  (1) Manual Navigator requires Adobe Reader.[3].

  (2) If Manuals folder is moved, Manual Navigator cannot show them.

2.10.3. Starting the MAP Viewer

- Startup

  You can start MAP Viewer using one of the following two methods.

  (1) To start MAP Viewer from the High-performance Embedded Workshop

  Please set up High-performance Embedded Workshop in the following procedure.

  (A) In the menu, click [Setup] -> [Customize] to display the Customize dialog box.

  (B) Click the Menu tab in the Customize dialog box.

  (C) Click the Add button to display the Add Tool dialog.

  (D) Specify the following in the Add Tool dialog.

Name	MAP Viewer (any name is acceptable)
Command	C:\Program Files\Renesas\Hew\Tools\Renesas\nc100\v101r00\bin\MapViewer.exe (specify the mapviewer.exe in the compiler install directory)
Arguments	$(CONFIGDIR)\$(PROJECTNAME).x30
Initial directory	$(CONFIGDIR)

  (2) To start Call Walker from Windows Start menu

  In All Programs of Windows Start menu, locate the Renesas menu labeled "R32C-100 Series C Compiler V.1.01 Release 00" and then click MAP Viewer in it.

- Termination

  Termination: Click [Exit] on the [File] menu.

[3] Adobe and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

2.10.4. Starting the Stack Analysis Tool (Call Walker)

- Startup

  You can start Call Walker using one of the following two methods.

  (1) To start Call Walker from the High-performance Embedded Workshop

  Click Renesas Call Walker on the Tool menu of the High-performance Embedded Workshop.

  (2) To start Call Walker from Windows Start menu

  In All Programs of Windows Start menu, locate the Renesas menu labeled "R32C-100 Series C Compiler V.1.01 Release 00" and then click Call Walker in it.

- Termination

  Termination: Click [Exit] on the [File] menu.

- Creating Input Files for Call Walker

  Use the .sni file creation tool named gensni to create the input files for Call Walker.

  The method for creating the input files for Call Walker differs depending on how the absolute module files (x30) are built.

  (1) When built in the High-performance Embedded Workshop

  When you build an x30 file, gensni is automatically executed.

  (2) When compiled, assembled and linked at the command prompt (or DOS prompt)

  Execute gensni at the command prompt (or DOS prompt).

```
[Example for executing gensni]
  c:\> gensni   -o   sample.sni   sample.x30
```

- Selecting an Input File for Call Walker

  To select an input file for Call Walker, click Import Stack File on the File menu of Call Walker and then select one in the Stack File window that is displayed.

2.11. Setting when compiler is used on DOS prompt and command prompt

The environment variable of the C compiler is set to setnc100.bat that exists in the installation directory of the C compiler. Please execute setnc100.bat when you use the compiler on the DOS prompt and the command prompt.

2.11.1. Environment Variables and Path

Environment variable	use
BIN100	Directory in which the C compiler execution files (e.g., *exe) are stored
INC100	Directory in which the standard include files of the C compiler are stored
LIB100	Directory in which the standard library files of the C compiler are stored
TMP100	Directory in which the temporary files generated by the C compiler are stored
path	Directory in which the C compiler execution files (e.g., *exe) are stored Select the directory for which you have access rights.

2.11.2. Batch File

A batch file named “setnc100.bat” will be generated in the directory in which you’ve installed the C compiler. This file has written in it the environment variables that the C compiler uses.

To use the C compiler from the DOS or the command prompt, execute setnc100.bat.

- Contents written in the batch file

```
REM ***** Environment variable for R32C Toolchains *****
SET BIN100=C:\Program Files\Renesas\Hew\Tools\Renesas\nc100\v101r00\BIN
SET BIN100=C:\Program Files\Renesas\Hew\Tools\Renesas\nc100\v101r00\LIB100
SET BIN100=C:\Program Files\Renesas\Hew\Tools\Renesas\nc100\v101r00\INC100
SET BIN100=C:\Program Files\Renesas\Hew\Tools\Renesas\nc100\v101r00\TMP
SET PATH=%BIN100%;%PATH%
```

3. Software version list of C Compiler Package V.1.01 Release 00

The following lists the software items and their versions include with C Compiler Package.

- nc100 V.2.00.05.000
- igen100 V.1.00.00.000
- cpp100 V.1.00.03.000
- ccom100 V.1.01.06.000
- aopt100 V.1.00.01.000
- as100 V.1.00.03.000
- mac100 V.1.00.01.000
- asp100 V.1.00.04.000
- psfp100 V.1.00.01.000
- ln100 V.1.00.02.000
- lb100 V.1.00.01.000
- lmc100 V.1.00.01.000
- abs100 V.1.00.01.000
- gensni V.1.00.00.002
- genmap V.1.00.00.000
- MapViewer V.3.01.02

4. Conformance with MISRA C Rule

4.1. Standard Function Library

In C-Source code of standard function library C Compiler Package, it is found that 52 rules[4] are against the MISRA C Rule NOTE, but these violations do not constitute a drawback to any operation.

4.1.1. Cause of Rule Violation

In C-Source code of standard function library C Compiler Package, the major causes for rule violation are as follows:

(3) C-Compiler specifications (near/far modifier, asm () function and #pragma)
(4) Declaration of function based on ANSI Standard
(5) The evaluation sequence in the conditional statement is not described explicitly, using a parenthesis.
(6) Implicit type conversion

4.1.2. Inspection No. running counter to the rule

The following are Inspection Nos. that run counter to the Rule:

1	12	13	14	18	21	22	28	34	35
36	37	38	39	43	44	45	46	48	49
50	54	55	56	57	58	59	60	61	62
65	69	70	71	72	76	77	82	83	85
99	101	103	104	105	110	111	115	118	119
121	124								

[4] These results were produced after inspection using MISRAC Rule Checker for M32C/90, M32C/80, M16C/80 Series.

4.2. C startup files

In C-Source code of C startup files Compiler Package, it is found that 6 rules[5] are against the MISRA C Rule NOTE, but these violations do not constitute a drawback to any operation.

4.2.1. Cause of Rule Violation

In C-Source code of standard function library C Compiler Package, the major causes for rule violation are as follows:

C-Compiler specifications (near/far modifier, asm () function and #pragma)

(7) Declaration of function based on ANSI Standard

4.2.2. Inspection No. running counter to the rule

The following are Inspection Nos. that run counter to the Rule:

22 45 99

4.3. SFR header files(Used in startup files)

In C-Source code of sfr header files Compiler Package, it is found that 6 rules[6] are against the MISRA C Rule NOTE, but these violations do not constitute a drawback to any operation.

(1) C-Compiler specifications (near/far modifier, asm () function and #pragma)

(2) Declaration of typedef

(3) Declaration of member of bitfield

4.3.1. Inspection No. running counter to the rule

The following are Inspection Nos. that run counter to the Rule:

13 14 99 110 111

4.4. Evaluation Environment

C Compiler	C Compiler Package for R32C/100 Series V.1.01 Release 00
Compile Option	-O -c –as100 "-DOPTI=0" -gnone -finfo -fNII -misra_all –misra_report $*.csv
MISRA C Checker	SQMlint V.1.03 Release 00

[5] These results were produced after inspection using MISRAC Rule Checker for M32C/90, M32C/80, M16C/80 Series.
[6] These results were produced after inspection using MISRAC Rule Checker for M32C/90, M32C/80, M16C/80 Series.

5. C language Startup Program

5.1. File composition of C language Startup Program

C language Startup Program contains the following 13 files.

(4) resetprg.c
Initializes the microcomputer.

(5) initsct.c
Initializes each section (by clearing them to 0 and transferring initial values).

(6) heap.c
Reserves storage for the heap area.

(7) fvector.c
Defines the fixed vector table.

(8) intprg.c
Declares the entry function for variable vector interrupts.

(9) firm.c
Reserves storage for the program and workspace areas used by firm of NSD as dummy areas when OnChipDebugger is selected.

(10) resetprg.h
Include does each header file for C language Startup Program.

(11) initsct.h
Contains statements for the processes (assembler macros) that initialize each section.
Please do not alter the file.

(12) heapdef.h
Initializes the heap area.

(13) cregdef.h
Declares the internal registers of the microcomputer.
Please do not alter the file.

(14) stackdef.h
Defines the stack size.

(15) vector.h
Defines the variable vector address.

(16) typedef.h
Declares each type by typedef.

5.2. Processing of C language startup program

5.2.1. resetprg.c

Initializes the microcomputer .

This file is necessary for C language Startup Program.

```
#include "resetprg.h"
////////////////////////////////////////////
// declare sfr register
DEF_SBREGISTER;

#pragma entry start
void start(void);
extern void initsct(void);
extern void _init(void);
void exit(void);
void main(void);

#pragma section program interrupt                                          → (1)
#pragma inline set_cpu()
void set_cpu(void)                                                         → (2)
{
        _isp_   = &(unsigned long)_istack_top;  // set interrupt stack pointer   → (3)
        _flg_   = 0x0080;                       // set flag register          → (4)
        _sp_    = &(unsigned long)_stack_top;   // set user stack pointer     → (5)
        _sb_    = (unsigned long *)0x400;       // 400H fixation (Do not change) → (6)
        _asm("  fset    b");
        _sb_    = (unsigned long *)0x400;
        _asm("  fclr    b");
        _intb_ = (unsigned long *)VECTOR_ADR;   // set variable vector's address → (7)
}

void start(void)
{
        set_cpu();              // initialize mcu                          → (8)
        initsct();              // initlalize each sections                → (9)
#ifdef __HEAP__
        heap_init();            // initialize heap                         → (10)
#endif
#ifdef __STANDARD_IO__
        _init();                // initialize standard I/O                 → (11)
#endif
        _fb_ = 0;               // initialize FB registe for debugger
        main( );                // call main routine                       → (12)

        exit( );                // infinite loop
}

void exit(int rc)
{
        while(1);
}
```

(1) The startup function is located in the interrupt section.

(2) Declares the function body of the CPU initialization function set_cpu().

(3) Initializes the interrupt stack pointer.

(4) Sets the U flag to 1 (stack pointer changed for the user stack).

(5) Initializes the user stack pointer.

(6) Sets the SB register to address 0x400 (which sets the start address of RAM).

(7) Sets the variable vector address in the INTB register. The VECTOR_ADR that defines the variable vector address is defined in vector.h. Note also that if the variable vector address is altered by a link option for section order under the HEW environment, resetprg.c must always be recompiled.

(8) Calls the CPU initialization function.
(9) Initializes each section (by clearing them to 0 and transferring initial values).
(10) Initializes the heap area.If memory management functions are used, call to this function must be enabled.
(11) Initializes the standard input/output device.If standard input/output functions are used, call to this function must be enabled.
(12) Calls the main function.

5.2.2. resetprg.h

Include does each header file for C language Startup Program.
This file is necessary for C language Startup Program.

5.2.3. initsct.c

Initializes each section (by clearing them to 0 and transferring initial values).
This file is necessary for C language Startup Program.

```
#include "initsct.h"
void initsct(void);

void initsct(void)
{
        sclear("bss_SB8", "data,align");
        sclear("bss_NEAR","data,align");                    → (1)
        sclear("bss_FAR", "data,align");
        sclear("bss_EXT", "data,align");

        /* clear bss for NSD */
        sclear("bss_MON1", "data,align");
        sclear("bss_MON2", "data,align");
        sclear("bss_MON3", "data,align");
        sclear("bss_MON4", "data,align");

        // when add new sections
        // bss_clear("new section's name");

        scopy("data_SB8", "data,align");
        scopy("data_NEAR","data,align");                    → (2)
        scopy("data_FAR", "data,align");
        scopy("data_EXT", "data,align");

        /* copy data section for NSD */
        scopy("data_MON1","data,align");
        scopy("data_MON2","data,align");
        scopy("data_MON3","data,align");
        scopy("data_MON4","data,align");
}
```

(1) sclear: Clears the bss section of the near area to zero.
If the bss section name is altered or a new bss section name is added using the #pragma SECTION bss feature, NE and NO must be altered or added in pairs.

```
sclear(“section name_NEAR,” “data.align”);
```

Example: When a section is added by #pragma section bss bss2, the following must be added to init.sct.c

```
sclear( “bss2_NEAR” , “data.align” );
```

(2) scopy: Transfers initial values to the data section of the near area.
If the data section name is altered or a new dada section name is added using the #pragma SECTION data feature, NE and NO must be altered or added in pairs.

```
scopy("section name_NEAR," "data.align");
```

Example: When a section is added by #pragma section data data2, the following must be added to initsct.c

```
sclear( "data2_NEAR" , "data.align" );
```

Supplement:

The initialization of the section not to be using do a comment out.

As a result, it is possible to do the reduction of the ROM size and start up processing speeding-up.

(1) When "#pragma SBDATA" is not used, the attribute of section base deletes the initialization of "SB8".

(2) When "#pragma EXTMEM" is not used, the attribute of section base deletes the initialization of "EXT".

(3) When "#pragma MONITOR[n]" is not used, the attribute of section base deletes the initialization of "MON[n]".

5.2.4. initsct.h

Contains statements for the processes (assembler macros) that initialize each section.

This file is necessary for C language Startup Program.

Please do not alter the file.

5.2.5. heap.c

Reserves storage for the heap area.

Only when memory management functions such as malloc are used.

```
#include "typedefine.h"
#include "heapdef.h"
#pragma SECTION    bss          heap                    → (1)

_UBYTE heap_area[__HEAPSIZE__];                         → (2)
```

(1) Locates the heap area in the heap_NEAR section.

(2) Reserves storage for the heap area by an amount equal to the size defined in __HEAPSIZE__.

5.2.6. heapdef.h

Initializes the heap area.

This file is necessary for memory management functions.

```
extern      _UBYTE _far * _mnext;
extern      _UDWORD _msize;
////////////////////////////////////////////
// It's size of heap
// When you want to change size of heap,
// please change this line.
// When you change this line,
// you must modify the value using hex character.

#ifndef __HEAPSIZE__
#define __HEAPSIZE__ 0x300
#endif
extern _UBYTE heap_area[__HEAPSIZE__];

#pragma inline heap_init()
void heap_init(void)
{
            _mnext = &heap_area[0];                    → (1)
            _msize = __HEAPSIZE__;                     → (2)
}
```

(1) Initializes the heap management area.

(2) Initializes the heap size.

5.2.7. fvector.c

Defines the fixed vector table.

This file is necessary for C language Startup Program.

```
#include "vector.h"
#pragma sectaddress          fvector,ROMDATA Fvectaddr    → (1)

////////////////////////////////////////////////////////////////////////////

#pragma interrupt/v _dummy_int          //udi              → (2)
#pragma interrupt/v _dummy_int          //over_flow
#pragma interrupt/v _dummy_int          //brki
#pragma interrupt/v   0xffffffff
#pragma interrupt/v   0xffffffff
#pragma interrupt/v _dummy_int          //wdt
#pragma interrupt/v _dummy_int
#pragma interrupt/v _dummy_int          //nmi
#pragma interrupt/v start                                → (3)

#pragma interrupt _dummy_int()
void _dummy_int(void){}
```

(1) Outputs the section and address of a fixed vector table.
This pragma is used exclusively for startup and cannot normally be used.

(2) Fills fixed vectors other than reset with a dummy function (_dummy_int).
This pragma is used exclusively for startup and cannot normally be used.

(3) Defines the entry function.
The function to be executed upon reset is registered in a fixed vector.

5.2.8. intprg.c

Declares the entry function for variable vector interrupts.
The content of this file depends on the MCU.

```
// BRK       (software int 0)
#pragma interrupt     _brk(vect=0)
void _brk(void){}

// vector 1 reserved

// uart5 trance/NACK(software int 2)                    → (1)
#pragma interrupt     _uart5_trance(vect=2)
void _uart5_trance(void){}

// uart5 receive/ACK (software int 3)
#pragma interrupt     _uart5_receive(vect=3)
void _uart5_receive(void){}

// uart6 trance/NACK              (software int 4)
#pragma interrupt     _uart6_trance(vect=4)
void _uart6_trance(void){}

// uart6 receive/ACK              (software int 5)
#pragma interrupt     _uart6_receive(vect=5)
void _uart6_receive(void){}

                :
          (Omission)
                :
```

(1) Declares the variable vector interrupt function.
The functions corresponding to each variable vector interrupt function are declared. A variable vector table is generated at the same time.

5.2.9. firm.c

Reserves storage for the program and workspace areas used by firm of NSD as dummy areas when OnChipDebugger is selected.
The content of this file is altered depending on the microcomputer type and selected OnChipDebugger.

```
#include "typedefine.h"
#pragma section bss FirmRam                                  → (1)
_UBYTE _workram[0x8];          // for Firmware's workram     → (2)
```

(1) Allocates the work ram area to be used by the NSD firmware in the FirmRam_NEAR section.
(2) Reserves storage for the work ram area by an amount equal to the size defined in __WORK_RAM__.

5.2.10. cregdef.h

Declares the internal registers of the microcomputer.
This file is necessary for C language Startup Program.
Please do not alter the file.

5.2.11. stackdef.h

Defines the stack size.

This file is necessary for C language Startup Program.

```
#ifndef __STACKSIZE__
#pragma STACKSIZE  0x300                                    → (1)
#else
#pragma STACKSIZE  __STACKSIZE__                            → (2)
#endif
#ifndef ISTACKSIZE
#pragma ISTACKSIZE 0x300                                    → (3)
#else
#pragma ISTACKSIZE __ISTACKSIZE__                           → (4)
#endif
extern _UINT _stack_top,_istack_top;
```

(1) Indicates the default size of the user stack.

(2) Outputs a user stack section and reserves storage for it.

(3) Indicates the default size of the interrupt stack.

(4) Outputs an interrupt stack section and reserves storage for it.

5.2.12. vector.h

Defines the variable vector address.

This file is necessary for C language Startup Program.

```
#define Fvectaddr       0xffffffdc                          → (1)
#ifndef VECTOR_ADR
#define VECTOR_ADR          0x0ffffd00                      → (2)
#endif
```

(1) Indicates the start address of a fixed vector table.

(2) Indicates the start address of a variable vector table.

If the start address of a variable vector table is changed, the address that is set in the INTB register in resetprg.c must also be changed at the same time.

5.2.13. typedefine.h

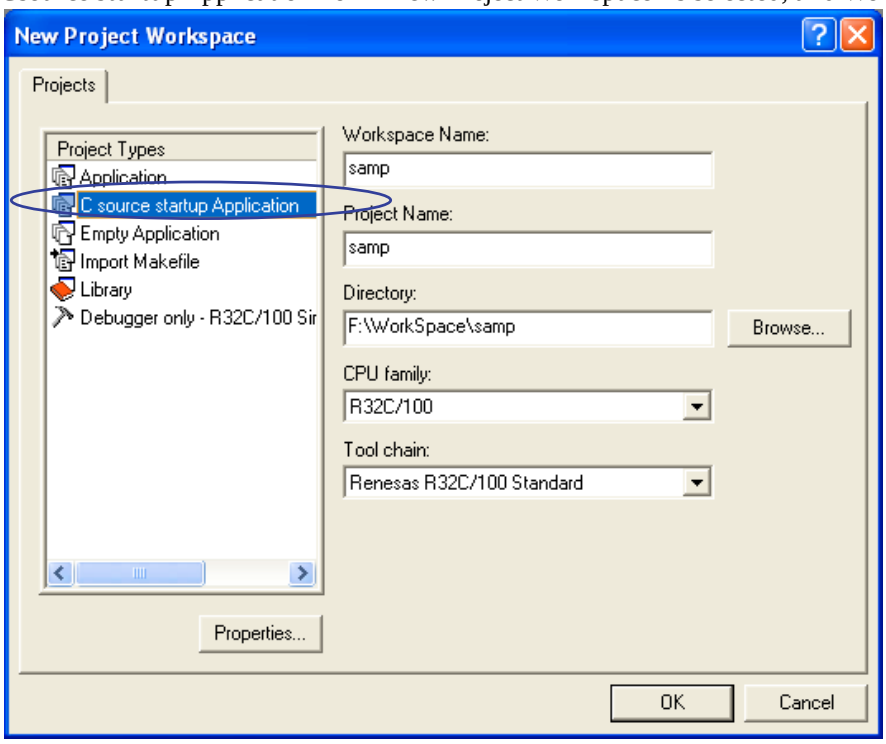
Declares each type by typedef.

This file is necessary for C language Startup Program.

Please do not alter the file.

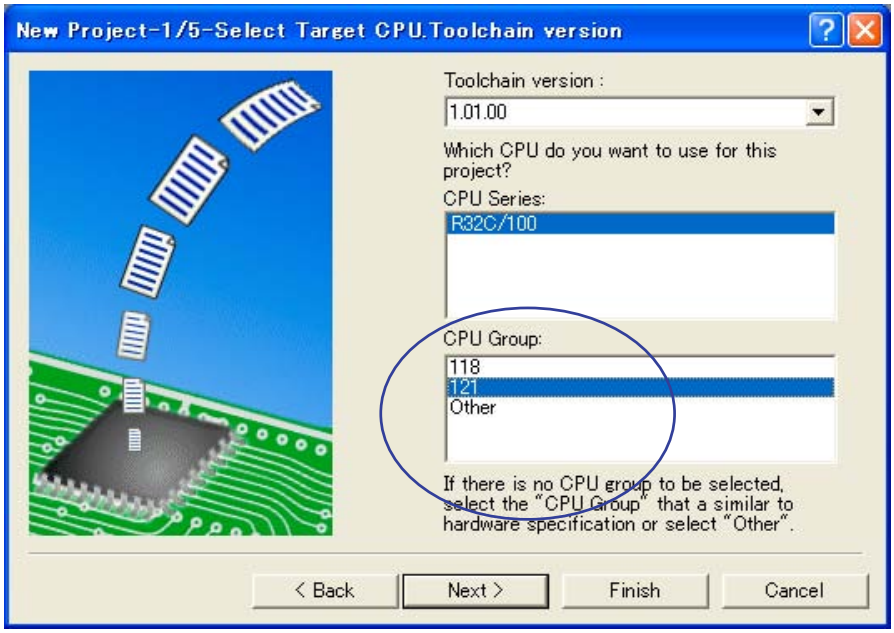
5.3. C language Startup Program is used on High-performance Embedded Workshop.

(1) “Csource startup Application “ of “New Project Workspace” is selected, and Workspace is made.



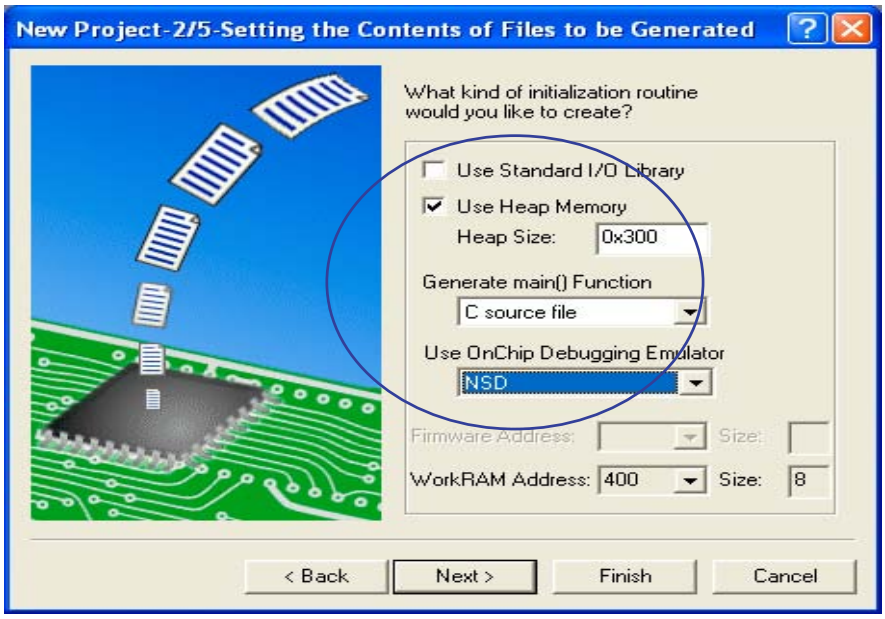
If while multiple compilers are installed in your computer you select another microcomputer for the CPU type after selecting C source startup Application, the focus for C source startup Application will move to Application, with the result that the selected C source startup has no effect. In such a case, therefore, select “C source startup Application” again.

(2) The target microcomputer is selected from "CPU Series" and "CPU Group".



When a type of microcomputer is selected, its corresponding sfr header file is copied to the workspace. Furthermore, a variable vector table (intprg.c) is registered.

(3) Settings for the case where the standard function and memory management function libraries are used

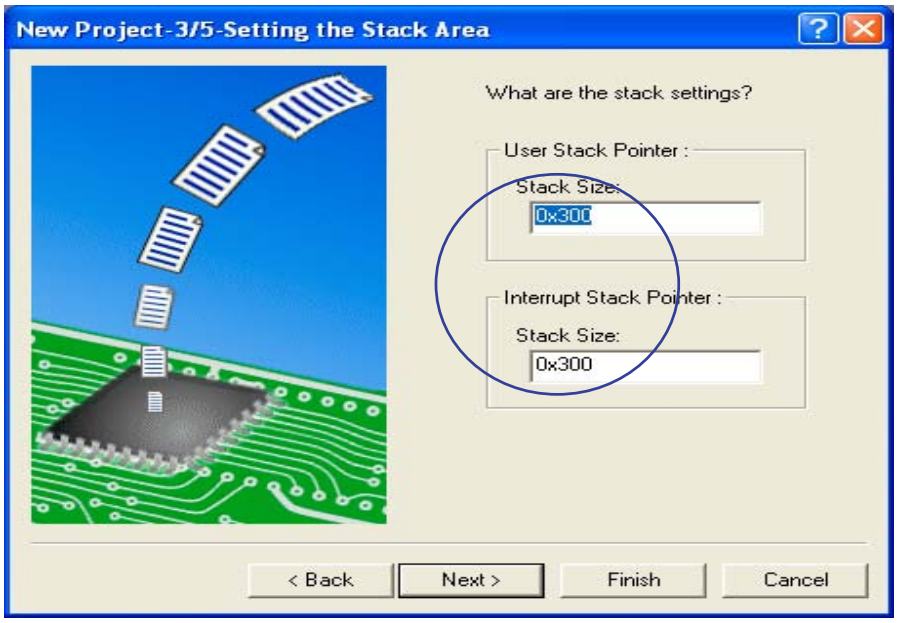


(A) Select this check box when you use the standard function library.
When this check box is selected (flagged with a check mark), function calls to _init() in resetprg.c are enabled.
Furthermore, device.c and init.c are registered to the project.

(B) Select this check box when you use the memory management function library.
When this check box is selected (flagged with a check mark), function calls to heap_init() in resetprg.c are enabled.
Furthermore, heapdef.h and heap.c are registered to the project.

(C) Select the appropriate debugger when you use OnChip Debugging Emulator.
The selectable debuggers are NSD.
Note, however, that you cannot select either one of the two or both depending on the selected type of microcomputer.
When this selection is made, firm.c is registered.

(4) Selecting the stack size



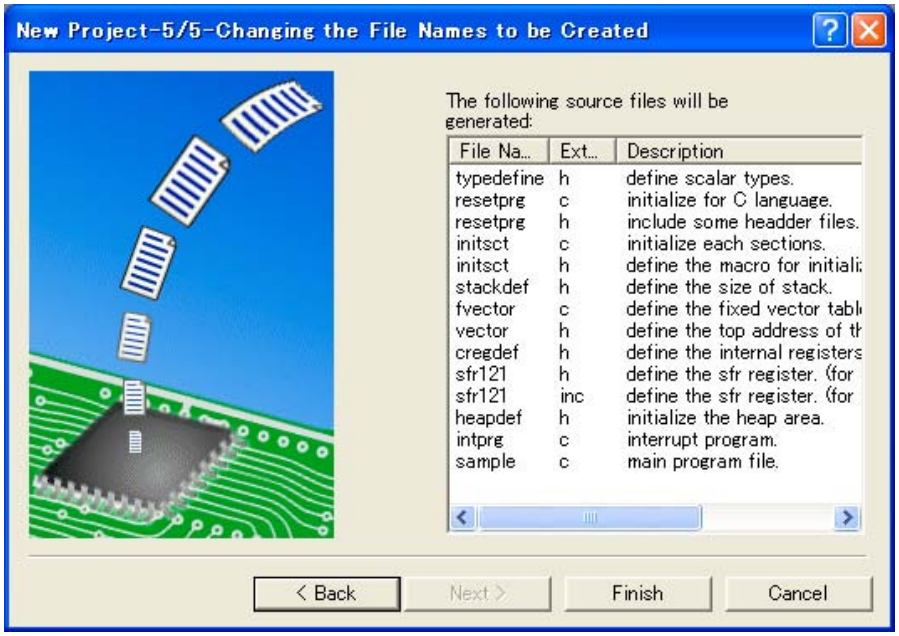
(A) Set the user stack size.
When this stack size is set, stackdef.h is registered.

(B) Set the interrupt stack size.
When this stack size is set, stackdef.h is registered.

To change the stack and HEAP sizes after creating a project, alter the value of each of the following in compile option settings:

```
-D__STACKSIZE__=xxxx
-D__ISTACKSIZE__=xxxx
-D__HEAPSIZE__=xxxx
```

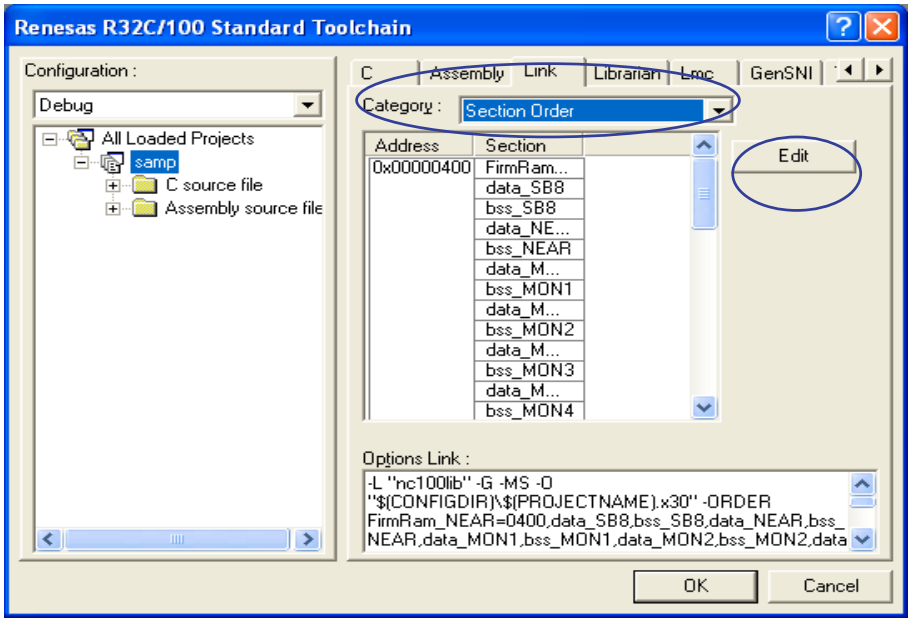
(5) List of registered files



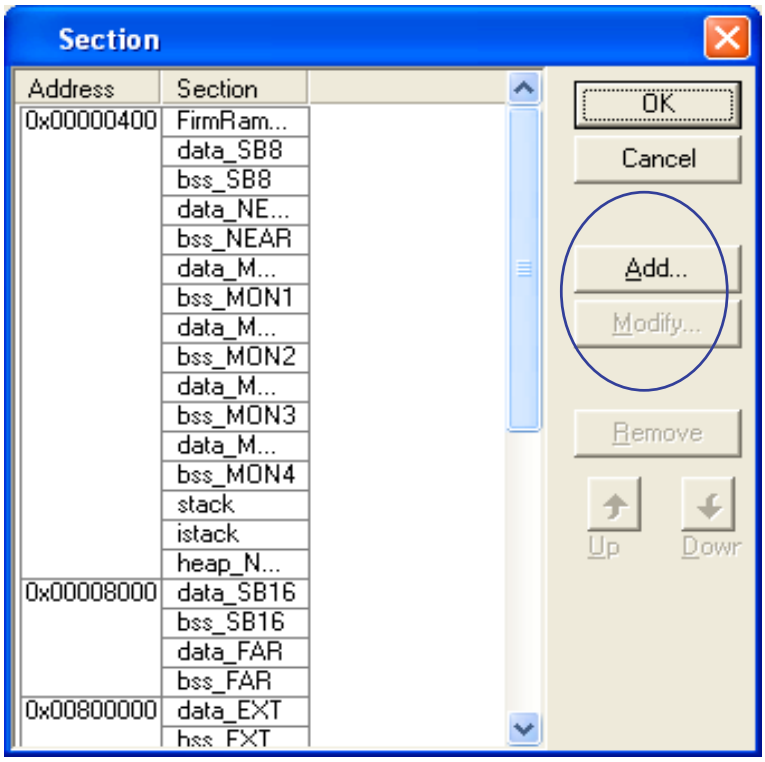
Here, you can check the list of files to be registered.
However, since the sfr header (C language header or assembler header) registered for each type of microcomputer is only copied to the workspace, take a look at this list to confirm the file name.

(6) Section Order

To confirm the order in which sections are linked and the addresses to which they are linked, take a look at "Category": Section Order in "Link" of "Renesas R32C/100 Standard Toolchain".



If you added a new section with #pragma SECTION, click the [Edit] button in (1) to open the Section window.



5.4. Assembly language Startup Program is used on High-performance Embedded Workshop.

"Application " of "New Project Workspace" is selected, and Workspace is made.