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

C Compiler Package for R32C/100 Series

V.1.02 Release 01

Release notes

(Rev.2.00)

Renesas Solutions Corporation

Jun 1, 2010

Abstract

Welcome to C Compiler Package for R32C/100 Series V.1.02 Release 01. 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. About Installation of C compiler Package

For details on how to install, please refer to “Install Guide”.

2. The latest information

Please refer to the following for the latest information on this product.
http://tool-support.renesas.com/eng/toolnews/p_r32c100.htm

3. Precautions on Product

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

3.1. Precautions about MS-Windows

3.1.1. Precautions about environment of operation

C Compiler Package operates under Windows 2000, Windows XP or Windows Vista. Is does not work under Windows 95, Windows 98, Windows ME, Windows NT 4.0 or earlier.

3.1.2. Precautions about MapViewer

As you cannot use Online Help of the MapViewer with a PC running Windows Vista, please use the Map Window of High-performance Embedded Workshop instead.

3.1.3. Suggestions Concerning File Names

The file names ,directory names and Workspace[1] names that can be specified are subject to the following restrictions:

(1) The directory, file, or workspace name which comprised of ASCII character-code only can 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.

If the limitations above are violated, the following problems may occur.

- The value set by the assembler directive commands .id, .ofsreg, .protect,.rvector or .svector cannot operate correctly. As a result, the ID code and the option function select register may not be set correctly.
- Call Walker and STK Viewer to refer to the stack size are not displayed correctly.
- The MAP Viewer to refer to the map information in the absolute module file isn't displayed correctly.
- The setting by these assembler directive commands isn't displayed in .map file.
- A compile error like "Can't open file" arises.
- A message like "Because a problem occurred, lnxx.exe is terminated." is issued and then the linker is terminated abnormally.
- The automatic generation function of the variable vector table is not be performed correctly.

3.1.4. 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.

[1] Workspace is a working directory used for processing like the compilation ,build ,or debugging on High-performance Embedded Workshop.

3.2. Precaution of MCU-Dependent Code

3.2.1. 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 */
}
```

- Workaround

(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 */
}
```

3.2.2. 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.

3.3. Precautions about C Compiler

3.3.1. About –Oglobal_jmp(-OGJ)

If the compiler option -Oglobal_jmp(-OGJ) , the assembler option –JOPT and the link option -JOPT are used and the link option -ORDER or –LOC is specified more than one time, only either -ORDER or -LOC that is specified last time becomes effective and a linkage error occurs.
As a result:

- If -ORDER is specified more than one time, a linkage error will occur.
- If -LOC is specified more than one time, allocation will not be done properly.

Please be sure to specify -ORDER and -LOC respectively one by one.

3.3.2. 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.

3.3.3. 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.

3.3.4. 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.

3.3.5. 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 ) ;
}
```

3.3.6. 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
}
```

3.3.7. Precautions about malloc() ,calloc() and realloc()

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

3.3.8. Note on Defining an Incomplete Structure or Union Type

When the definition of a member follows that of an incomplete structure or union type (in which only a tag has been defined) using typedef, the members in structures or unions that are declared with the typedef name may not be displayed by any debuggers.
Note, however, that debuggers will display these members correctly when their definition precedes typedef.

- Example of occurrence

```
typedef struct str1    str1_t;     /* An incomplete structure type is defined */

struct str1 {                      /* Members are defined */
      int i;
      int j;
};

str1_t    s = { 1, 2 };            /* A structure is declared with the typedef name */
```

4. Contents of upgrade from V.1.02 Release 00

4.1. Fixed Problems

The following known problems have been fixed

- With accessing arrays successively
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/100116/tn1.htm
- With Errors Arising after Linking is Performed
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/091116/tn3.htm
- With calculating stack usage
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/091116/tn2.htm
- With the function for automatically generating variable vector tables
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/091001/tn4.htm
- With using SQMlint (the MISRA C rule checker) with High-performance Embedded Workshop
  [RENESAS TOOL NEWS]
  http://tool-support.renesas.com/eng/toolnews/090801/tn1.htm

5. Startup or termination of program

5.1. 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.

5.2. 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.[2].
  (2) If Manuals folder is moved, Manual Navigator cannot show them.

5.3. 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\v102r01\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.1.02 Release 00" and then click MAP Viewer in it.
- Termination
  Termination: Click [Exit] on the [File] menu.

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

5.4. 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.1.02 Release 01" 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.

5.5. 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.

5.5.1. 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.

5.5.2. 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\v102r01\BIN
SET BIN100=C:\Program Files\Renesas\Hew\Tools\Renesas\nc100\v102r01\LIB100
SET BIN100=C:\Program Files\Renesas\Hew\Tools\Renesas\nc100\v102r01\INC100
SET BIN100=C:\Program Files\Renesas\Hew\Tools\Renesas\nc100\v102r01\TMP
SET PATH=%BIN100%;%PATH%
```

6. Software version list of C Compiler Package V.1.02 Release 01

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

- nc100 V.2.00.06.002
- igen100 V.1.00.00.000
- cpp100 V.1.01.00.000
- ccom100 V.1.02.02.001
- aopt100 V.1.01.02.001
- as100 V.1.00.04.001
- mac100 V.1.00.01.000
- asp100 V.1.01.00.000
- psfp100 V.1.00.01.000
- ln100 V.1.02.00.001
- lb100 V.1.00.01.000
- lmc100 V.1.01.00.000
- abs100 V.1.00.01.000
- gensni V.1.00.00.002
- genmap V.1.00.01.001
- MapViewer V.3.01.02

7. Conformance with MISRA C Rule

7.1. Standard Function Library

In C-Source code of standard function library C Compiler Package, it is found that 52 rules[3] are against the MISRA C Rule NOTE, but these violations do not constitute a drawback to any operation.

7.1.1. Cause of Rule Violation

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

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

7.1.2. 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								

[3] These results were produced after inspection using MISRAC Rule Checker SQMLint.

7.2. C startup files

In C-Source code of C startup files Compiler Package, it is found that 6 rules[4] are against the MISRA C Rule NOTE, but these violations do not constitute a drawback to any operation.

7.2.1. Cause of Rule Violation

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

(1) C-Compiler specifications (near/far modifier, asm () function and #pragma)
(2) Declaration of function based on ANSI Standard

7.2.2. Inspection No. running counter to the rule

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

22 45 99

7.2.3. #pragma extended functions for use in C startup (Misra C rule 99)

Extended Function	Definition File	Description	Function
#pragma STACKSIZE	stackdef.h	Defines the user stack size.	The stack section (stack) is output and the top label name of the stack is generated.
#pragma ISTACKSIZE	stackdef.h	Defines the interrupt stack size.	The interrupt stack section (istack) is output and the top label name of the interrupt stack is generated.
#pragma CREG	cregdef.h	Declares an internal register of the MCU.	A special instruction is used to generate code for access to an internal register declared by this pragma.
#pragma sectaddress	fvector.c	Defines a section. Its address can also be declared at the same time.	The section name declared by this pragma is used to define a section.. When its address is specified at the same time, an address definition using a pseudo instructions ".org" is output.
#pragma entry	resetprg.c	Declares a function to be executed at the time of a reset.	An enter instruction to configure a stack frame for the function declared by this pragma is not output. This is because the enter instruction should not be generated before the stack pointer is initialized.
#pragma interrupt/v	fvector.c	Generates a vector table.	Only the interrupt vector is defined for the function declared by this pragma.
#pragma inline	heapdef.h resetprg.c	Declares an inline function.	The function declared by this pragma is inline-expended.
#pragma interrupt	intprg.c fvector.c	Declares an interrupt function.	Interrupt-function code is generated for the function declared by this pragma.
#pragma section	heap.c resetprg.c initsct.h firm.c	Changes the name of a section.	The section name is changed to the one defined by this pragma.
#pragma ADDRESS	Each sfr header file	Defines the I/O address and declares a variable.	.equ is used to define the I/O address for sfr defined by this pragma.

[4] These results were produced after inspection using MISRAC Rule Checker SQMLint.

7.3. SFR header files(Used in startup files)

In C-Source code of sfr header 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.

7.3.1. Cause of Rule Violation

In C-Source code of sfr header files C Compiler Package, the major causes for rule violation are as follows:

(1) C-Compiler specifications (near/far modifier, asm () function and #pragma)
(2) Declaration of typedef
(3) Declaration of member of bitfield

7.3.2. Inspection No. running counter to the rule

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

13 14 99 110 111

7.4. 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 SQMLint.

8. C language Startup Program

8.1. File composition of C language Startup Program

C language Startup Program contains the following 13 files.

(1) resetprg.c
Initializes the microcomputer.

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

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

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

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

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

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

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

(9) heapdef.h
Initializes the heap area.

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

(11) stackdef.h
Defines the stack size.

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

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

8.2. Processing of C language startup program

8.2.1. 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) Defines 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.

8.2.2. resetprg.h

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

8.2.3. 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, NEAR and FAR 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, NEAR and FAR 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]".

8.2.4. 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.

8.2.5. 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__.

8.2.6. 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.

8.2.7. 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.

8.2.8. 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.

8.2.9. 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__.

8.2.10. cregdef.h

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

8.2.11. 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.

8.2.12. 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          0x0fffffd00                     → (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.

8.2.13. typedefine.h

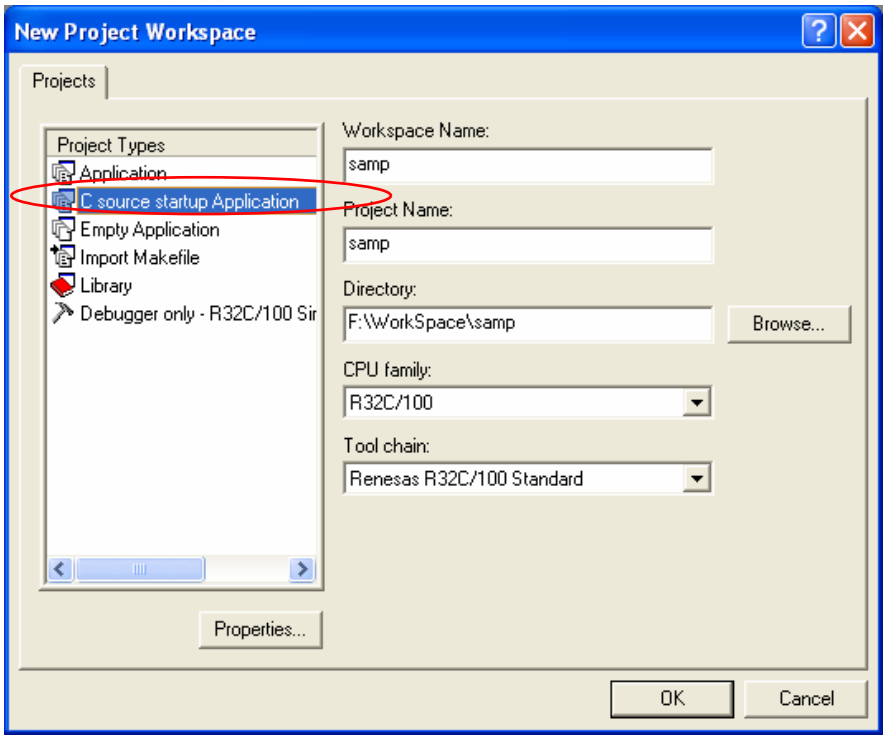
Declares each type by typedef.

This file is necessary for C language Startup Program.

Please do not alter the file.

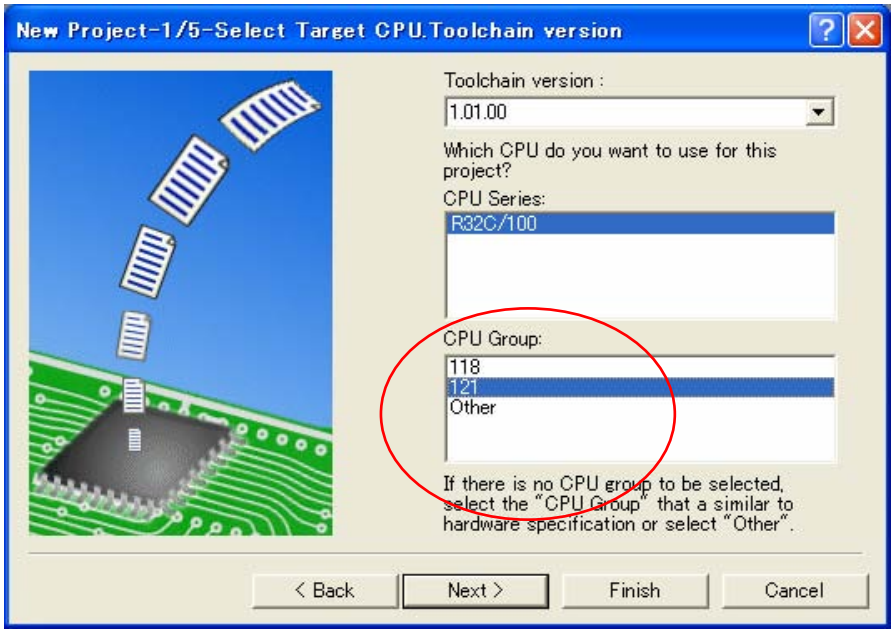
8.3. C language Startup Program is used on High-performance Embedded Workshop.

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



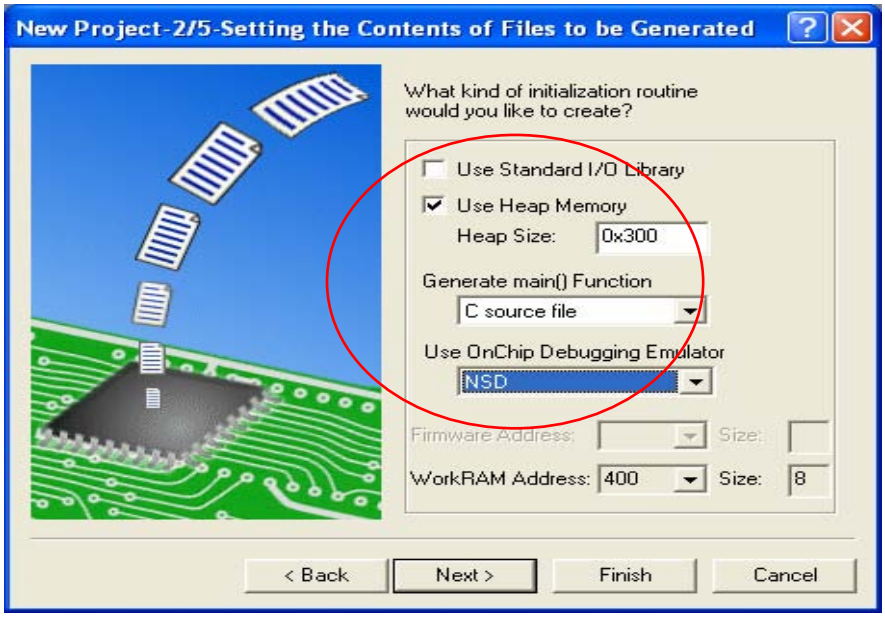
If while multiple compilers are installed in your computer you select another microcomputer for the CPU type after selecting C source startup Application, the focus for C source startup Application will move to Application, with the result that the selected C source startup has no effect. In such a case, 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 I/O function and memory management function libraries are used

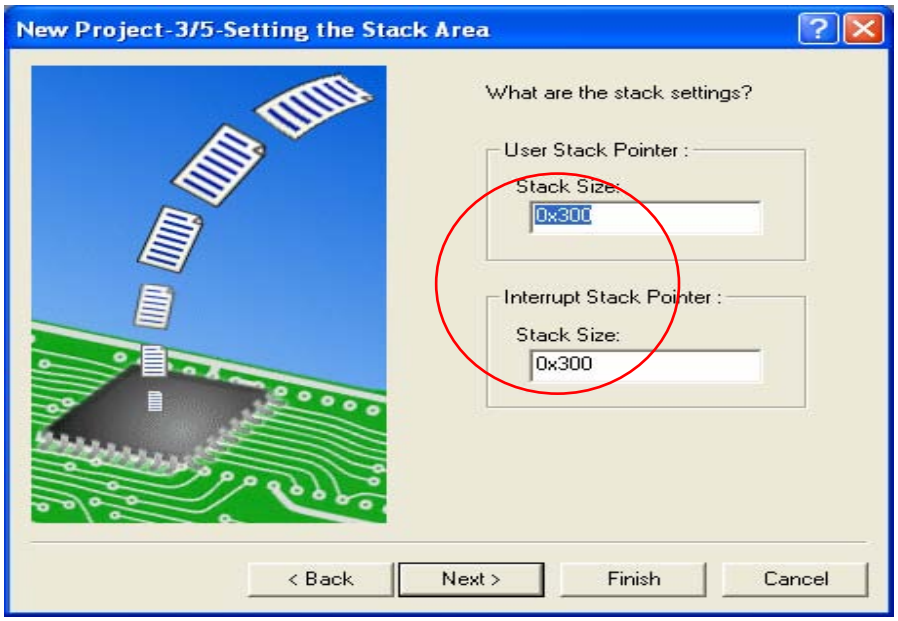


(A) Select this check box when you use the standard I/O function library.
When this check box is selected (flagged with a check mark), function calls to _init() in resetprg.c are enabled.
Furthermore, device.c and init.c are registered to the project.

(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



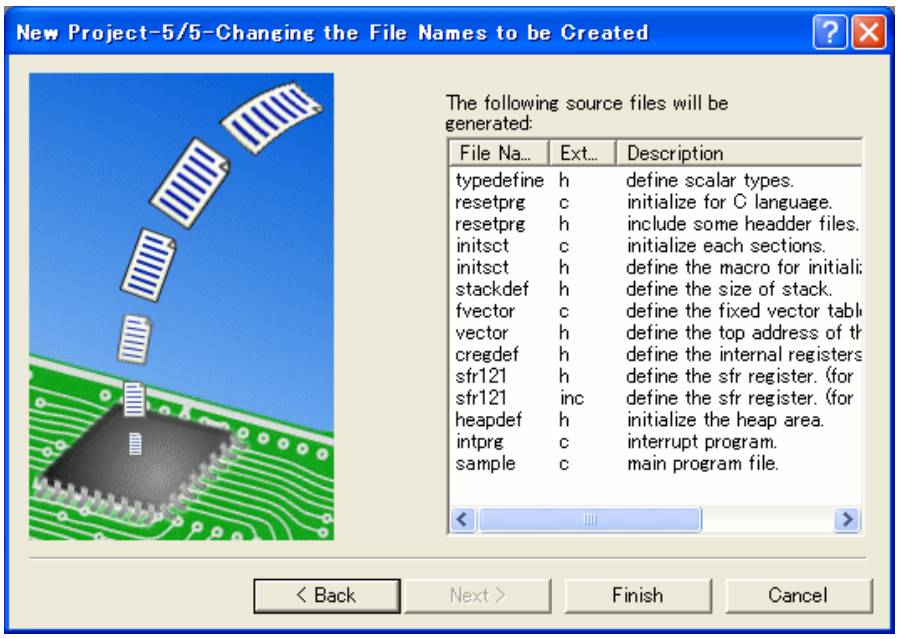
(A) Set the user stack size.
When this stack size is set, stackdef.h is registered.

(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
```

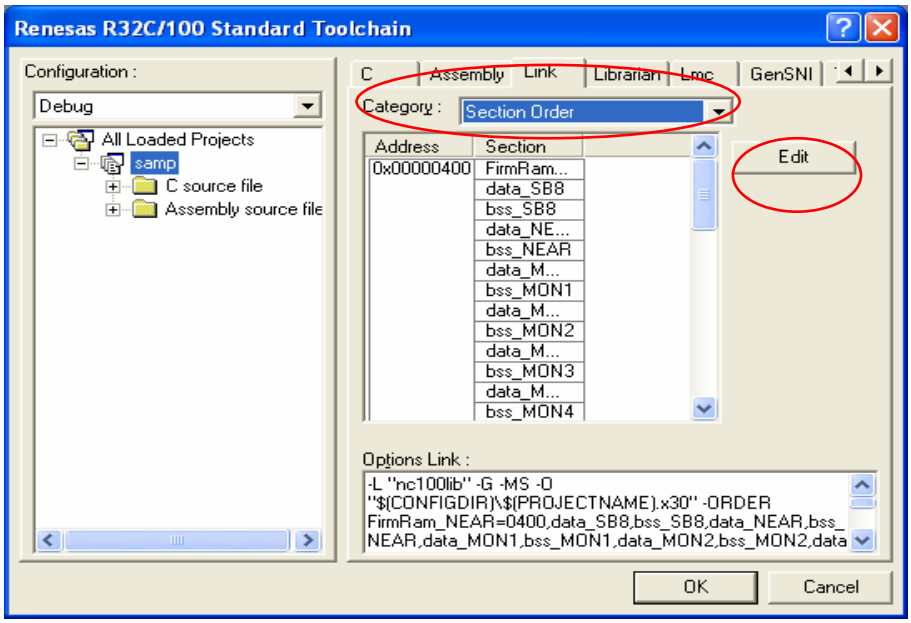
(5) List of registered files



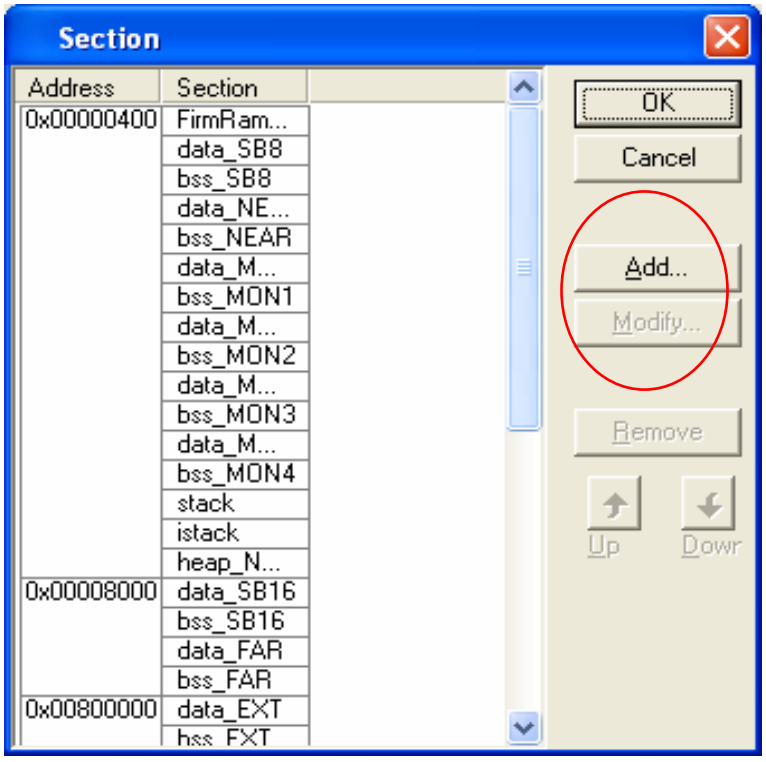
Here, you can check the list of files to be registered.
However, since the sfr header (C language header or assembler header) registered for each type of microcomputer is only copied to the workspace, take a look at this list to confirm the file name.

(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.



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

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